John-Harry Wieken

MASTER CLASS

SQL

» Einstieg für Anspruchsvolle

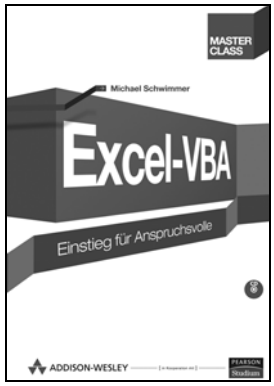
inkl. Lösungen und Lerntest, Beispiele

ADDISON-WESLEY [in Kooperation mit] PEARSON Studium

SQL

MASTER CLASS

Hier eine Auswahl:



Excel-VBA

Michael Schwimmer
512 Seiten
€ 29,95 [D], € 30,80 [A]
ISBN 978-3-8273-2525-9

In diesem Buch erfahren Sie die Dinge, die den Einsteiger vom Profi unterscheiden. Alle wichtigen Themen für das professionelle Programmieren mit Excel-VBA werden ausführlich angesprochen. Wichtige Codebestandteile werden farblich gekennzeichnet. Auf Übersichtlichkeit und didaktische Führung wurde besonderer Wert gelegt. Mithilfe des Testprogramms auf CD kann das erworbene Wissen zusätzlich überprüft werden.

Access-VBA

Stefan Leibing, Bernd Held
480 Seiten
€ 29,95 [D], € 30,80 [A]
ISBN 978-3-8273-2264-7

Visual Basic für Applikationen ist eine der bekanntesten prozeduralen Programmiersprachen. Mit VBA lassen sich Office-Anwendungen wie Access, Word oder Excel programmieren. Seine große Flexibilität verdankt es den zahlreichen Anwendungen, die eigene Schnittstellen für den Zugriff mit VBA bereitstellen. Das Buch liefert dem Leser einen qualifizierten Einstieg in VBA mit Access und enthält alle Beispieldaten zum Nachvollziehen und den Lerntest auf CD.

John-Harry Wieken

SQL

» Einstieg für Anspruchsvolle

ADDISON-WESLEY [in Kooperation mit] PEARSON Studium

Bibliografische Information Der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

Die Informationen in diesem Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht ausgeschlossen werden.
Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt. Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das ® Symbol in diesem Buch nicht verwendet.

Umwelthinweis:
Dieses Produkt wurde auf chlorfrei gebleichtem Papier gedruckt. Um Rohstoffe zu sparen, haben wir auf Folienverpackung verzichtet.

10 9 8 7 6 5 4 3 2 1

11 10 09

ISBN 978-3-8273-2485-6

© 2009 Pearson Studium,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten
www.pearson-studium.de
Lektorat: Brigitte Bauer-Schiewek, bbauer@pearson.de
Fachlektorat: Dirk Louis
Herstellung: Martha Kürzl-Harrison, mkuerzl@pearson.de
Korrektorat: Martina Gradias
Coverkonzeption und -gestaltung: Marco Lindenbeck, webwo GmbH
(mlindenbeck@webwo.de)
Satz: Reemers Publishing Services GmbH, Krefeld (www.reemers.de)
Druck und Verarbeitung: Kösel, Krugzell (www.KoeselBuch.de)

Printed in Germany

Auf einen Blick

Vorwort 13

Kapitel 1 Einleitung 15

Kapitel 2 SQL – der Standard relationaler Datenbanken 19

Kapitel 3 Die Beispieldatenbanken 35

Kapitel 4 Mit SQL Daten abfragen (SELECT) 89

Kapitel 5 Datentypen, Ausdrücke und Funktionen 149

Kapitel 6 Datenbankinhalte ändern (INSERT, UPDATE, DELETE) 199

Kapitel 7 Datenbanken modellieren 219

Kapitel 8 Datenbanken erstellen (SQL-DDL) 249

Kapitel 9 Unterabfragen (Sub-SELECT) 301

Kapitel 10 Unterabfragen in der DDL und DML 317

Kapitel 11 Mengenoperationen (UNION, INTERSECT, EXCEPT/MINUS) 327

Kapitel 12 Benutzer, Rechte und Zugriffsschutz 339

Kapitel 13 Transaktionen 357

Kapitel 14 Mit SQL Datenbanken betreiben und optimieren . . 369

A Anhang: Benutzung der Datenbanksysteme 385

B Anhang: Boolesche Alge#bra 413

C Anhang: Daten 417

Stichwortverzeichnis 423

Inhaltsverzeichnis

Vorwort 13

Kapitel 1 Einleitung 15

Kapitel 2 SQL – der Standard relationaler Datenbanken ... 19

2.1 Die Geschichte 19
2.2 Die Bestandteile 20
2.3 Die Verarbeitung einer SQL-Anweisung 22
2.4 Die Struktur von SQL-Anweisungen 25
2.5 Relationale Datenbanken 29
2.5.1 Tabellen 29
2.5.2 Primärschlüssel 32
2.5.3 Beziehungen 33

Kapitel 3 Die Beispieldatenbanken 35

3.1 Die Kursdatenbank 35
3.2 Schnelleinstieg und Neustart 37
3.2.1 MySQL 38
3.2.2 Oracle 38
3.2.3 Firebird 39
3.2.4 MS Access 40
3.2.5 openBase 40
3.3 MySQL 41
3.3.1 Gründe für die Nutzung 41
3.3.2 Den Server installieren und konfigurieren 41
3.3.3 Die Kommandozeile 51
3.3.4 Die grafische Oberfläche MySQL Tools 54
3.3.5 Eine Beispieldatenbank aufbauen 58
3.4 MS Access 65
3.4.1 Gründe für die Nutzung 65
3.4.2 Eine Beispieldatenbank aufbauen 66
3.5 Oracle 69
3.5.1 Gründe für die Nutzung 69
3.5.2 Oracle installieren 69
3.5.3 Die Testdatenbanken importieren 72
3.6 Firebird 74
3.7 OpenOffice.orgBase 82

Kapitel 4 Mit SQL Daten abfragen (SELECT) 89

4.1 SELECT – die Syntax 89
4.2 Einfache Abfragen 91
4.3 Daten sortieren mit der ORDER BY-Klausel 100
4.4 Die Daten mit der WHERE-Klausel auswählen 104
4.5 Tabellen miteinander verbinden (JOIN) 113
4.5.1 Der Klassiker (INNER JOIN) 113
4.5.2 JOIN über mehrere Tabellen 119
4.5.3 Varianten des INNER JOIN 123
4.5.4 Non-Equi-JOIN 125
4.5.5 OUTER JOIN 126
4.5.6 CROSS JOIN 130
4.5.7 JOIN über mehrere Felder 133
4.6 Die GROUP BY-Klausel 134
4.7 Die HAVING-Klausel 144

Kapitel 5 Datentypen, Ausdrücke und Funktionen 149

5.1 Datentypen 149
5.1.1 Alphanumerische Angaben (Text) 153
5.1.2 Ganze Zahlen 156
5.1.3 Gleitkommazahlen 158
5.1.4 Datum/Uhrzeit 160
5.1.5 BITs, BLOBs und andere Datentypen 162
5.2 NULL-Werte 163
5.3 Literale 164
5.4 Ausdrücke 167
5.5 Funktionen 168
5.6 Datensatzorientierte Funktionen (Skalarfunktionen) 171
5.6.1 Funktionen in MS Access 172
5.6.2 Numerische Funktionen 174
5.6.3 Alphanumerische Funktionen 180
5.6.4 Datumsorientierte Funktionen 183
5.6.5 Datentypumwandlungsfunktionen (Casting) 187
5.6.6 Logische und sonstige Funktionen 190
5.7 Gruppenorientierte Funktionen (Aggregatfunktionen) 193

Kapitel 6 Datenbankinhalte ändern (INSERT, UPDATE, DELETE) 199

6.1 Neue Datensätze einfügen (INSERT) 199
6.1.1 INSERT mit Werten und Funktionen 199
6.1.2 INSERT mit Unterabfragen 202
6.1.3 INSERT mit SET 205

6.1.4 Besonderheiten des INSERT mit MS Access ... 205
6.1.5 INSERT-Übungen ... 209
6.2 Vorhandene Datensätze ändern (UPDATE) ... 210
6.2.1 UPDATE-Anweisungen ... 211
6.2.2 Besonderheiten von UPDATE bei MS Access ... 213
6.2.3 Zusammenfassung ... 214
6.2.4 Update-Übungen ... 215
6.3 Datensätze löschen (DELETE) ... 215
6.3.1 DELETE-Grundlagen ... 215
6.3.2 Alle Datensätze löschen (TRUNCATE) ... 216
6.3.3 Besonderheiten des DELETE bei MS Acess ... 217
6.3.4 Übungen zur DELETE-Anweisung ... 218

Kapitel 7 Datenbanken modellieren ... 219

7.1 Das 3-Ebenen-Modell ... 219
7.1.1 Anforderungen an das Datenbankmodell ... 219
7.1.2 Die drei Ebenen des Datenbankmodells 220
7.1.3 Der Weg zum Datenbankmodell ... 222
7.2 Das Entity-Relationship-Modell (ERM) ... 224
7.2.1 Entitäten ... 226
7.2.2 Attribute (Eigenschaften) ... 226
7.2.3 Domänen ... 227
7.2.4 Beziehungen ... 228
7.3 Beispiel BüroFix ... 232
7.4 Umsetzung in das relationale Modell ... 237
7.5 Sinn und Unsinn der Normalisierung ... 240
7.5.1 Redundanz und Anomalien ... 240
7.5.2 Normalisierungsziele ... 242
7.5.3 Funktionale Abhängigkeit ... 243
7.5.4 Normalformen ... 245
7.5.5 Grenzen der Normalisierung ... 246

Kapitel 8 Datenbanken erstellen (SQL-DDL) ... 249

8.1 Das Datenbankschema erstellen (CREATE SCHEMA) ... 249
8.1.1 MySQL ... 250
8.1.2 MS Access ... 253
8.1.3 Oracle ... 254
8.1.4 Firebird ... 255
8.1.5 openBase ... 256
8.1.6 Übungen ... 257

8.2 Tabellen erstellen (CREATE TABLE) 257
8.2.1 Standardangaben für Felder 257
8.2.2 Fremdschlüsselbeziehungen 263
8.3 Integritätsbedingung 266
8.3.1 Primärschlüssel (PRIMARY KEY) 266
8.3.2 Fremdschlüssel erstellen (FOREIGN KEY) 268
8.3.3 Allgemeine Integritätsbedingung (CHECK) 275
8.3.4 UNIQUE-Bedingung 277
8.3.5 Übungen zu Integritätsbedingungen ... 278
8.3.6 MS Access 279
8.4 Die Tabellen ändern (ALTER TABLE) 281
8.5 Tabellen löschen (DROP TABLE) 284
8.6 Benutzer und Programmsichten (CREATE VIEW) 285
8.6.1 Spaltenselektion 287
8.6.2 Zeilenselektion 288
8.6.3 Tabellen kombinieren 289
8.6.4 Der VIEW in MySQL 291
8.6.5 Der VIEW in MS Access 293
8.6.6 Einen VIEW ändern (ALTER VIEW, DROP VIEW) 294
8.6.7 Änderbarkeit eines VIEW 294
8.6.8 Übungen 296
8.7 Domänen 297
8.7.1 Domänen erstellen 297
8.7.2 Domänen ändern (ALTER DOMAIN) ... 299
8.7.3 Domänen löschen (DROP DOMAIN) ... 299
8.7.4 Übungen 300

Kapitel 9 Unterabfragen (Sub-SELECT) 301

9.1 Nutzung von Unterabfragen 301
9.2 Unterabfragen mit Vergleichsoperatoren 305
9.3 Unterabfragen mit ALL und ANY 307
9.4 Unterabfragen mit IN und EXISTS 309
9.5 Synchronisierte und korrelierte Unterabfragen . 312
9.6 Regeln für Unterabfragen in der WHERE-Klausel 313
9.7 Erweiterungen der Unterabfragen 313
9.8 Unterabfragen mit MS Access 315

Kapitel 10 Unterabfragen in der DDL und DML 317

10.1 CREATE mit Unterabfragen 317
10.2 UPDATE mit Unterabfragen 320
10.3 INSERT mit Unterabfragen 321
10.4 DELETE mit Unterabfragen 323

Kapitel 11 Mengenoperationen (UNION, INTERSECT, EXCEPT/MINUS) 327

11.1 Überblick 327
11.2 Die Vereinigungsmenge (UNION) 328
11.3 Die Schnittmenge (INTERSECT) 333
11.4 Die Differenzmenge (MINUS/EXCEPT) 334
11.5 Besonderheiten der Datenbanksysteme 335
11.6 Zusammenfassung 337
11.7 Übungen 337

Kapitel 12 Benutzer, Rechte und Zugriffsschutz 339

12.1 Schutz der Informationen 339
12.2 Benutzer und Benutzergruppen 340
12.3 Benutzer entfernen (DROP USER) 342
12.4 Rollen einrichten 342
12.5 Rollen löschen 342
12.6 Rechte einrichten (GRANT) 343
12.7 Rechte entziehen (REVOKE) 344
12.8 Benutzerkonzepte verschiedener Datenbanken 345
12.8.1 Benutzerkonzept in MySQL 345
12.8.2 Benutzerkonzept in Oracle 347
12.8.3 Benutzerkonzept in Firebird 350
12.8.4 Benutzerkonzept in MS Access 352
12.8.5 Benutzerkonzept in openBase 353
12.9 VIEW als Zugriffsschutz 354

Kapitel 13 Transaktionen 357

13.1 Grundlagen 357
13.2 AUTOCOMMIT 359
13.3 Eigenschaften einer Transaktion 360
13.4 Sperren 361
13.5 Transaktionen und Benutzervariablen 367

Kapitel 14 Mit SQL Datenbanken betreiben und optimieren 369

14.1 Optimierter Zugriff – der INDEX 369
14.1.1 Nutzen von Indizes 369
14.1.2 Einen Index anlegen (CREATE INDEX) 370
14.2 Einen Index löschen 375

14.3 Weitere Überlegungen zum Einsatz von Indizes ... 375
14.4 Weitere Anweisungen zur physischen Datenspeicherung ... 379
14.5 Prozeduren und Trigger ... 380
14.6 Application Program Interface ... 381
14.7 Abschluss ... 383

A Anhang: Benutzung der Datenbanksysteme ... 385

A.1 MySQL ... 385
A.2 Oracle ... 389
A.3 Firebird ... 394
A.4 MS Access ... 396
1.4.1 SQL-Anweisungen eingeben ... 396
A.4.2 Die Daten aus Excel importieren ... 401
A.5 openBase ... 407
A.5.1 SQL-Anweisungen eingeben ... 407
A.5.2 Andere SQL-Anweisungen eingeben ... 411

B Anhang: Boolesche Algebra ... 413

C Anhang: Daten ... 417

C.1 Datenbank Kurse ... 417
C.2 Datenbank Artikel ... 419

Stichwortverzeichnis ... 423

Vorwort

Dieses Buch möchte Ihnen einen Weg zu SQL zeigen und dabei stets den Nutzen im Rahmen der Datenbankverwendung und -pflege in den Vordergrund stellen. Ich werde die wesentlichen Aspekte von SQL vor dem Hintergrund der Prinzipien relationaler Datenbanken erläutern und dabei immer auf die unmittelbare Umsetzung eingehen. Das Buch beruht auf SQL92 mit den wesentlichen Erweiterungen von SQL99, SQL2003 und SQL2006.

Meine berufliche wie pädagogische Praxis hat mir gezeigt, dass jedes Lernen von der Übung lebt. Wie man das Tennisspielen nicht aus dem Fernsehen lernen kann, kann man das Programmieren nicht durch das Lesen eines Buches erlernen. Also beginnen die Abschnitte fast immer mit einem Beispiel und enden mit Übungsaufgaben. Dazwischen werden die wesentlichen Aspekte erläutert, mit Beispielen versehen und schließlich noch zusammen mit der Syntax der SQL-Anweisung zusammengefasst.

SQL ist eine universelle Datenbanksprache, die allerdings Dialekte aufweist. Dialekte, die von den unterschiedlichen Datenbanksystemen gesprochen werden. Mit den Datenbanken ist es wie mit den Dialekten, es gibt kein Besser oder Schlechter, man mag einen Dialekt oder man mag ihn nicht. Schließlich hängt es auch davon ab, wo man lebt, genauso wie von dem Datenbanksystem, das man lernen möchte, vielleicht weil es in der Firma eingesetzt wird, der Kunde es wünscht, die Hochschule oder Schule es vorgibt oder man es einfach schon ein wenig kennt. Vielleicht auch, weil man ein neues System kennenlernen möchte.

Ich möchte Sie herzlich einladen, Ihre Entscheidung frei zu treffen und vielleicht auch mit mehreren Systemen parallel zu experimentieren. Es soll Ihnen hier kein Dialekt vorgeschrieben werden. Daher sind alle Lösungen zu den Übungsaufgaben und viele Hinweise im Text auf die Systeme MySQL, MS Access, Oracle, Firebird und openOffice.base (hier kurz: openBase) bezogen. Basis für meine Auswahl war, dass diese Systeme zusammen die in der Praxis verwendeten Systeme gut repräsentieren und von allen (bis auf MS Access) zumindest kostenlose Übungsversionen verfügbar sind, die Sie auch auf der CD finden. Wählen Sie also Ihr System, Ihren „Dialekt“ oder, um im obigen Bild zu bleiben, Ihren „Tennisschläger“ zum Erlernen von SQL.

Mein Dank gilt zunächst dem Verlag Addison-Wesley für die Geduld und die vielen fleißigen Hände, die es letztlich ermöglicht haben, dieses Buch zu verlegen und zu Ihnen zu bringen. Der Dank gilt insbesondere Frau Bauer-Schiewek für ihre Geduld, ihre stets schnelle Reaktion und die gute Betreuung. Besonderer Dank gilt Herrn Dirk Louis für viele Anregungen, Verbesserungsvorschläge und Korrekturen.

Für dieses Buch habe ich eine eigene Internetseite eingerichtet: *www.serval.de/SQL*. Ich werde versuchen, dort zeitnah Anregungen zu kommentieren, Fehler zu korrigieren und sonstige Informationen auszutauschen.

John-Harry Wieken, Celle

1

Einleitung

Die zentrale Bedeutung von SQL liegt in seiner fast monopolartigen Position beim Zugriff auf relationale Datenbanken. Diese wiederum bilden das Rückgrat der Datenverarbeitung von großen, mittleren und immer mehr auch kleineren Unternehmen. Eine Bank, eine Versicherung oder ein Industriebetrieb ohne eine relationale Datenbank ist schwer vorstellbar. Spätestens dann, wenn mehr als ein Mitarbeiter gleichzeitig auf den Datenbestand zugreifen soll, ist eine Datenbank fast unerlässlich. Das kann beispielsweise ein Warenwirtschaftssystem, eine Internetpräsenz oder ein Berichtssystem sein. Datenbanken verbergen sich hinter Excel-Anwendungen genauso wie hinter klassischen maskenorientierten Systemen, hinter Web-Oberflächen wie hinter einer Großrechneranwendung.

Datenbanken spielen also eine zentrale Rolle für die Datenverarbeitung. Relationale Systeme haben in den vergangenen zehn Jahren ihre vorherrschende Rolle weiter ausgebaut und sind heute fast konkurrenzlos. Dies liegt neben der hohen Flexibilität des relationalen Modells auch und gerade in der Möglichkeit eines standardisierten Zugriffs auf alle diese Systeme, der auch einen Namen trägt: SQL.

SQL – dieses unscheinbare Kürzel, bei dem bis heute der Streit darüber anhält, ob es „Standard Query Language“ oder „Structured Query Language“ heißt. Richtig ist übrigens historisch der zweite Begriff. Praktisch beschreibt die erste Variante die Situation als Standard.

SQL, das bedeutet eine einzige Sprache, um auf alle wichtigen Datenbanksysteme zuzugreifen, sei es DB2, ORACLE, Adabas, SQL-Server, MySQL, MS Access, openBase, MaxDB, Postgres oder Firebird (Interbase), um nur einige zu nennen. Darüber hinaus gibt es aus fast allen Programmiersprachen Zugriffsmöglichkeiten auf SQL. Es gibt objektorientierte Erweiterungen, Erweiterungen für XML und standardisierte Schnittstellen wie ODBC, JDBC oder DAO und ADO.

Datenbanken werden mit SQL definiert, mit Daten versorgt und abgefragt. Zugriffsmechanismen werden gesetzt, Optimierungen vorgenommen und die Datenorganisation wird gesteuert.

Wir wollen uns hier mit den verschiedenen Aspekten von SQL beschäftigen. Sie müssen dazu nicht die Schlagworte und Abkürzungen kennen, die hier gefallen sind. Wir wollen hier ganz vorsichtig anfangen und mit verständlichen Begriffen Schritt für Schritt die Möglichkeiten von SQL erschließen. Dementsprechend ist das Buch so aufgebaut, dass jedes Kapitel ein eigenes Modul darstellt. Sollten Sie bereits Vorkenntnisse haben, können Sie natürlich die entsprechenden Abschnitte überspringen oder schnell überfliegen.

Aufbau des Buches

Das Buch selbst gliedert sich in fünf Hauptabschnitte mit zusammen 14 Kapiteln. Die Struktur soll es Ihnen ermöglichen, einzelne Abschnitte nach Bedarf zu lesen, ohne gezwungen zu sein, alles von vorn bis hinten zu bearbeiten. Die folgenden Zeilen sollen ein kleiner Wegweiser durch das Buch sein.

Abschnitt I
Kapitel 1 – 3
Vorbereitung

Kapitel 2 dient dem Überblick über SQL und ist jedem Leser empfohlen, der ein paar Hintergrundinformationen zur Geschichte und dem Grundaufbau von SQL haben möchten. Kapitel 3 dient dazu, Ihnen die Umgebung der Übungsbeispiele nahezubringen. Die Beispieldatenbanken werden vorgestellt und die Installation der Beispielsysteme beschrieben. Die Beispiele und Übungen sind wie erwähnt für die SQL-Datenbankmanagementsysteme MySQL, MS Access, Oracle, openBase und Firebird erarbeitet, sodass Sie selbst Ihr bevorzugtes System auswählen können. Die Windows-Versionen sind teilweise auf der mitgelieferten CD. Zu allen Versionen erhalten Sie den Download oder entsprechende Hinweise auf aktualisierte Versionen auf *www.serval.de/SQL*.

Abschnitt II
Kapitel 4 – 6
Abfrage und Bearbeitung

Kapitel 4 ist der schnelle Einstieg für den Datenbanknutzer. Wir wollen mithilfe der SELECT-Anweisung Daten aus fertigen Beispieldatenbanken abfragen, Informationen zusammenstellen und analysieren. Dies ist die Kernfunktionalität aller Datenbanknutzer. Entsprechend wird der SELECT-Anweisung hier breiter Raum eingeräumt. Kapitel 5 vertieft die Nutzung der SELECT-Anweisung um Berechnungen und Umwandlungen der Ergebnisse mithilfe von SQL-Funktionen. Basis sind die SQL-Datentypen. Kapitel 6 führt in die Änderung (UPDATE), die Eingabe (INSERT) und das Löschen (DELETE) von Daten mit SQL ein. Jetzt können wir also die Daten in der Datenbank beliebig ändern.

Abschnitt III
Kapitel 7 – 8
Erstellen einer Datenbank

Kapitel 7 beschreibt die Entwicklung von Datenbanken mit dem konzeptionellen ER-Modell und zeigt dessen Umsetzung in ein relationales Modell mit Normalisierung. Kapitel 8 nimmt das so erstellte relationale Modell auf und führt dies fort, indem die SQL-Anweisungen zur Erstellung (CREATE), Änderung (ALTER) und Löschung (DROP) von Datenbankstrukturen und nicht „nur“ von Daten vorgestellt werden. Dieser Abschnitt umfasst also das Vorgehen zur Erstellung neuer Datenbanken.

Kapitel 9 nimmt das Thema „Abfragen“ jetzt auf der Basis der ursprünglichen Datenbanken wie auch der im vorherigen Abschnitt erstellten neuen Datenbanken wieder auf und zeigt die Nutzung sogenannter Unterabfragen (Sub-Query) im Zusammenhang mit der `SELECT`-Anweisung. Die Unterabfragen werden anschließend in Kapitel 10 auch auf die `INSERT`-, `UPDATE`-, `DELETE`- und `CREATE`-Anweisungen übertragen. Abschließend wird in Kapitel 11 auf die Mengenoperationen zur Kombination mehrerer `SELECT`-Anweisungen eingegangen.

Abschnitt IV
Kapitel 9 – 11
Komplexe Abfragen

Abschließend werden im letzten Abschnitt verschiedene Aspekte des Betriebs von Datenbanken angesprochen. Kapitel 12 geht auf die Verwaltung von Benutzern und Benutzergruppen sowie das Berechtigungskonzept ein. Kapitel 13 beschreibt die Sicherung der Datenkonsistenz eines und mehrerer paralleler Benutzer über das Konzept der Transaktionen. Kapitel 14 geht abschließend auf einige Aspekte von SQL ein, die die physische Datenspeicherung betreffen.

Abschnitt V
Kapitel 12 – 14
Aspekte des Betriebs

2

SQL – der Standard relationaler Datenbanken

2.1 Die Geschichte

Die Grundlagen der Datenbanksprache SQL gehen bereits auf den Anfang der Siebzigerjahre des vorigen Jahrhunderts zurück. Im Jahr 1974 stellte IBM erstmals eine Sprache mit dem Namen SEQUEL (Structured English Query Language) vor, den Urahnen unseres heutigen SQL. Wenn man in den USA über SQL spricht, wird man immer wieder feststellen, dass viele Amerikaner bis heute SQL als „Siquel“ aussprechen, wenn auch natürlich die direkte Aussprache der Abkürzung SQL („Es-Kju-El“) ebenfalls häufig zu hören ist.

Mathematisches Modell

SQL steht für „Structured Query Language“, also auf Deutsch etwa „strukturierte Abfragesprache“. Zum Zeitpunkt ihrer Entstehung war SQL eine von mehreren Sprachen, mit denen man versuchte, einen möglichst komfortablen Zugang zu den damals ebenfalls neuen relationalen Datenbanksystemen zu schaffen. E.F. Codd hatte 1970 das relationale Datenbankmodell vorgestellt. Seiner extrem einfachen, auf wenigen – aber klaren – mathematischen Regeln beruhenden Struktur verdankt es seinen Siegeszug, den es seitdem in der Informatik angetreten hat. Umgangssprachlich ausgedrückt beruht das relationale Modell im Wesentlichen auf der Mengenlehre.

Alle Informationen werden entsprechend ihrem Inhalt in Tabellen aufgeteilt und dort strukturiert in Datensätzen (auch Tupel genannt) gespeichert. Die Tabellen sind untereinander über Beziehungen (auch Relationen genannt) verbunden, die die einfache Kombination von Informationen aus verschiedenen Tabellen erlauben, also beispielsweise, welcher Kunde welche Bestellung abgegeben hat. Das System hat sich als extrem flexibel und zugleich einfach zu verwalten herausgestellt, sodass im Lauf der letzten 30 Jahre immer mehr Unternehmen ihre Datenspeicherung auf relationale Datenbanken umgestellt haben.

Abfragesprachen

Mit der Entwicklung der relationalen Datenbanken als Speicherform für praktisch jede Art strukturierter Informationen stellte sich die Frage nach dem Zugriff und der Verarbeitung dieser Informationen. Man suchte nach möglichst einfachen, zugleich aber umfassenden Wegen, um die Informationen in relationalen Datenbanken zu verarbeiten. Zahlreiche Namen wie QBE (Query by Example), QUEL (Query Language) und letztlich auch SEQUEL spiegeln diese verschiedenen Entwicklungen wider.

Durchgesetzt hat sich am Ende SEQUEL in seinem inzwischen mehrfach standardisierten Nachfolger SQL. Die Gründe für den Erfolg sind vielfältig. So ist die Marktmacht von IBM (und später deren Absplitterung Oracle) gerade im Bereich größerer Unternehmen nicht zu unterschätzen. Diese war gerade zum Entstehungszeitpunkt sehr groß. Heute ist SQL zum Selbstgänger geworden. Kein Datenbankanbieter von IBM über Oracle, Microsoft bis zu den OpenSource-Anbietern wie MySQL kann ohne den Standard SQL auskommen. Kein Anbieter von Standardsoftware oder Programmierumgebungen kann ohne eine Zugriffsschicht für SQL auskommen. Schließlich haben auch in heutigen Webanwendungen SQL-Datenbanken – insbesondere MySQL – ihren festen Platz gefunden. Neben der Unterstützung durch wichtige Hersteller von Software im Unternehmensbereich ist SQL auch dank seiner Standardisierung ein zentraler Pfeiler der Softwarearchitektur geworden.

ANSI-Standard

Die Standardisierung von SQL wird vom American National Standard Institute (ANSI) betrieben. Entsprechend spricht man auch von ANSI-Standards. Im Lauf der Jahre wurden verschiedene Standards für SQL verabschiedet, die entsprechend des jeweiligen Jahres mit SQL86, SQL89, SQL92, SQL99, SQL2003 und SQL2006 bezeichnet werden. SQL92 wurde auch als SQL2 und SQL99 als SQL3 bezeichnet. Die meisten verfügbaren realen Datenbanken stützen sich immer noch auf SQL92 und haben die nachfolgenden Standards in unterschiedlichem Umfang in ihrem „Dialekt“ implementiert.

2.2 Die Bestandteile

Die Gründe für den Erfolg von SQL sind neben der Unterstützung durch namhafte Anbieter und der Standardisierung auch in der Sprache SQL selbst zu suchen. SQL bietet eine sehr einfache Syntax, die konsequent an der Idee der relationalen Datenbanken orientiert ist. SQL-Anweisungen sind vergleichsweise leicht zu schreiben, zu lesen oder mit Programmgeneratoren zu erzeugen. Gerade der letzte Umstand ist nicht zu vernachlässigen, werden doch sehr viele SQL-Anweisungen nicht manuell, sondern maschinell erstellt. Doch auch maschinell erstellte Anweisungen müssen unter Umständen durch Menschen analysiert werden, sei es um Fehler zu finden, sei es um die Performance der Abfragen zu erhöhen oder auch um die Datenbankstruktur für bestimmte SQL-Abfragen zu optimieren.

SQL-Bestandteile

SQL bietet einen kompletten Satz von Anweisungen, um mit relationalen Datenbanken umgehen zu können. Insbesondere sind hier zu erwähnen:

- SQL-DDL (Data Definition Language): Die DDL dient der Erstellung, Änderung und Löschung von Datenbankstrukturen. Hiermit werden also die grundlegenden Strukturen der Datenbank (Tabellen, Felder, Views, ...) verändert.
- SQL-DML (Data Manipulation Language): Die DML dient der Abfrage, dem Einfügen, der Änderung und dem Löschen von Daten in bereits vorhandenen Datenbankstrukturen, also vorhandenen Tabellen, Feldern und Beziehungen.
- SQL-DCL (Data Control Language): Die DCL dient der Pflege der Datenbankinfrastruktur, insbesondere der Zugriffsberechtigungen.

Während die SQL-DDL und die SQL-DCL vorwiegend von Datenbankadministratoren verwendet werden, ist die SQL-DML das wesentliche Instrument für den Datenbankanwender, sei es ein Mensch oder ein Programm. Die SQL-DML lässt sich aufgrund der Aufgaben noch einmal grob in zwei Gruppen untergliedern:

- Die rein lesende Abfrage von Daten (SELECT)
- Die Veränderung der Daten durch das Einfügen, Ändern und Löschen (INSERT, UPDATE, DELETE)

Von herausragender Bedeutung ist dabei insbesondere das Lesen von Informationen mithilfe von SELECT, das dem Anwender die ganze Flexibilität der Datengewinnung und Analyse zur Verfügung stellt, die relationale Datenbanken bieten. In Kapitel 4 wird daher ausführlich auf den grundsätzlichen Aufbau des SELECT eingegangen.

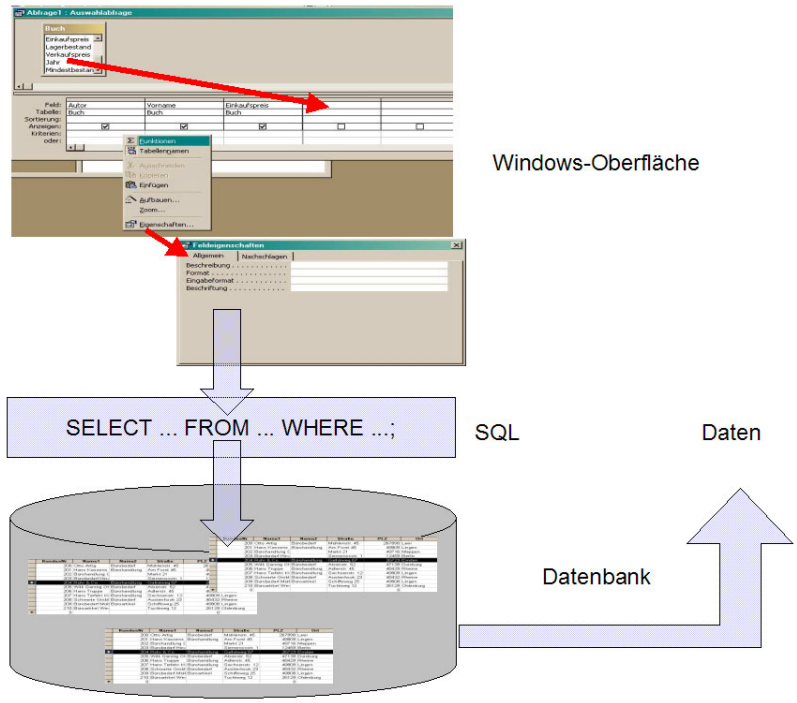
Programmiersprache

SQL ist recht einfach zu lesen und zu verstehen, trotzdem bleibt es eine Programmiersprache. Die Mächtigkeit von SQL ist beeindruckend, man muss sie allerdings auch beherrschen. Wir wollen uns in diesem Buch ausführlich mit der Sprache beschäftigen, denn wie bei jeder (Programmier-)Sprache kommt es auf die Syntax an. Die Wörter müssen sinnvoll und syntaktisch richtig kombiniert werden. SQL ist keine Windows-orientierte Sprache oder Anwendung. Helfen können bei der Erstellung von SQL-Anweisungen Programme mit grafischer Oberfläche, in denen SQL-Anweisungen „per Mausklick“ erstellt und generiert werden. Doch auch wenn Sie diese Programme nutzen, sollten Sie in der Lage sein, die erzeugten SQL-Anweisungen zu lesen und zu verstehen – sei es zum Zweck der Fehlersuche, der Einbindung in Programme oder der Performanceanalyse.

MS Access ist nur ein Beispiel für eine solche Oberfläche. MS Access wurde in diesem Buch unter anderem deswegen berücksichtigt, um eine derartige Oberfläche beispielhaft zeigen und den Zusammenhang zwischen der Oberfläche und dem eigentlichen SQL demonstrieren zu können. Schließlich haben viele Hersteller von Datenbanken zusätzliche Oberflächen zur Generierung von SQL entwickelt, um dem Anwender entgegenzukommen, der nicht programmieren möchte. Diese Oberflächen haben alle gemeinsam, dass

in einer Windows-typischen grafischen Umgebung die gewünschte Abfrage von Informationen aus der Datenbank zusammengestellt wird. Der Anwender beschreibt dabei die Eigenschaften der gewünschten Informationen in einem Formular. Hat der Anwender die gewünschten Informationen beschrieben, kann er sie per Knopfdruck aus der Datenbank erhalten. Tatsächlich werden aus den Beschreibungen des Anwenders SQL-Anweisungen generiert. Die Informationen werden dann mit diesen SQL-Anweisungen aus der Datenbank gewonnen und dem Anwender zur Verfügung gestellt. Das Vorgehen ist in Abbildung 2.1 schematisch dargestellt.

Abbildung 2.1
Generierung von SQL-Anweisungen über eine Windows-Oberfläche



2.3 Die Verarbeitung einer SQL-Anweisung

Die Verarbeitung einer SQL-Anweisung ist in Abbildung 2.1 eher anschaulich dargestellt.

Data Dictionary

Technisch sind in die Verarbeitung eine ganze Reihe von Komponenten eingebunden, die Abbildung 2.2 etwas detaillierter wiedergibt. Der große Rahmen umfasst die Komponenten des Datenbanksystems im engeren Sinne. Die Datenbank beinhaltet dabei sämtliche Informationen über ihre eigene Struktur, die Strukturen, die Anwender angelegt haben, also Tabellen, Beziehungen, Benutzer und Benutzergruppen, Berechtigungen und viele weitere Informationen, die der Verarbeitung dienen. Diese Informationen sind im Data Dictionary gespeichert, das die Datenbank selbst verwaltet.

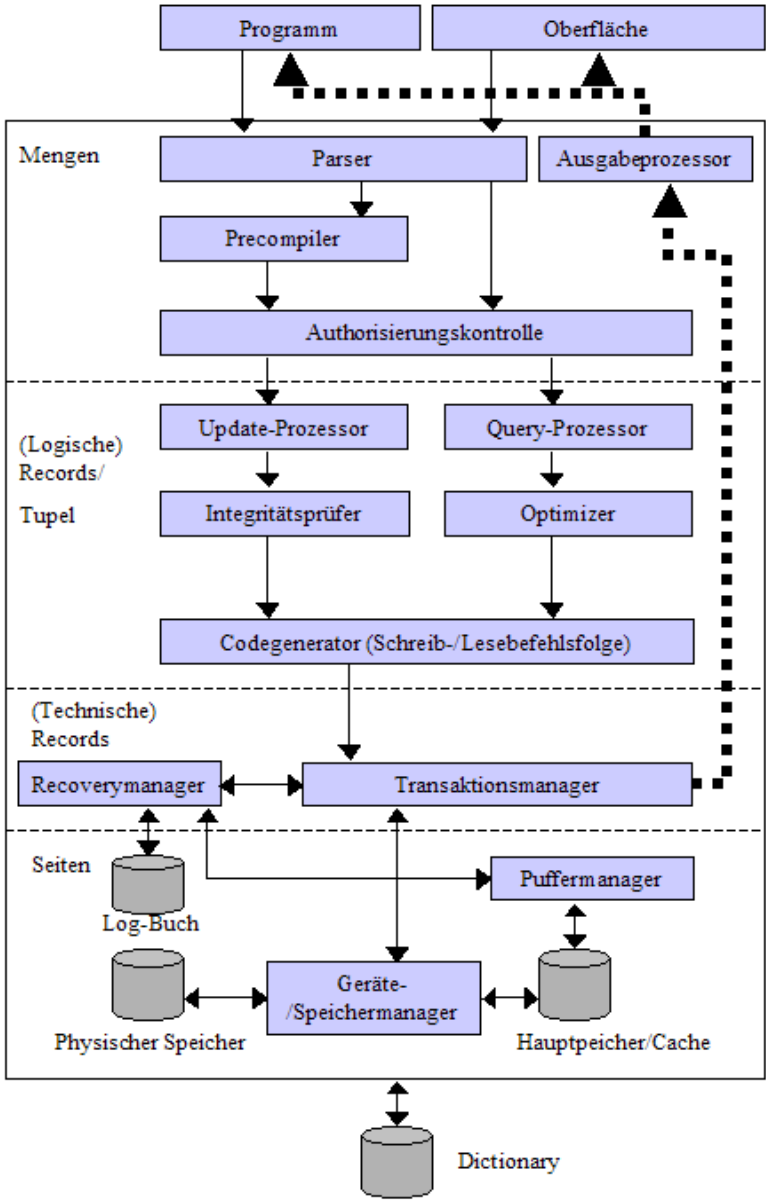


Abbildung 2.2
Grundablauf bei der SQL-Anweisung

Das Data Dictionary soll uns im Folgenden nicht so sehr beschäftigen, sondern nur als „Merkposten“ dienen, wenn unklar ist, wo bestimmte Informationen, die wir eingeben, bleiben, liegen beziehungsweise woher die Datenbank sie kennt. So werden beispielsweise alle Informationen, die bei der Erzeugung einer Datenbank mit SQL-Anweisungen (CREATE) angegeben werden (siehe Kapitel 8), im Data Dictionary gespeichert.

Programm oder Oberfläche

SQL-Befehle können prinzipiell auf zwei Wegen an die Datenbank gelangen. Zum einen über eine Programmierschnittstelle, bei der die SQL-Anweisungen in eine andere Programmiersprache wie C++, C#, Java, Pascal, COBOL, Basic, PHP, Perl, Ruby oder eine der vielen anderen Sprachen, für die Schnittstellen existieren, eingebettet werden. Zum anderen können die Anweisungen direkt über eine mitgelieferte und je nach Datenbanksystem mehr oder weniger grafische Oberfläche eingegeben werden. Zudem gibt es für praktisch alle Datenbanken weitere Oberflächenwerkzeuge, die von Dritten entwickelt und kostenfrei oder gegen Lizenzgebühr einen komfortableren Umgang mit der Datenbank ermöglichen. Wichtig ist, dass sowohl die Programme als auch die Oberflächen aus Sicht der Datenbank einen Benutzer darstellen, der SQL-Anweisungen an die Datenbank schickt und dafür einen Returncode und eventuell eine Datenmenge zurückgeliefert bekommt. Das ist die Aufgabe von SQL.

Parser

Nimmt die Datenbank eine SQL-Anweisung entgegen, wird diese zunächst auf formale Richtigkeit hinsichtlich der SQL-Syntax geprüft. Dies ist die Aufgabe des Parsers. Aufgabe dieses Buches ist es, dass Sie möglichst wenig mit dessen Fehlermeldungen zu tun bekommen, indem wir richtiges, für den Parser verständliches SQL verwenden. Dies bildet den Schwerpunkt dieses Buches. Verglichen mit der deutschen Sprache würde der Parser den Satz „Es heute regnen." als syntaktisch falsch abweisen. Die Kontrolle durch den Parser ist aber noch keine Garantie für eine sinnvolle SQL-Anweisung. Der Satz „Ich regne heute." ist in der deutschen Sprache syntaktisch richtig. Der Parser beurteilt also nicht den Inhalt der Anweisung. Ob eine Anweisung auch semantisch sinnvoll ist, sehen wir erst, wenn der Ausgabeprozessor die ermittelte Datenmenge und den Returncode aufbereitet hat.

Autorisierung

Eine vom Parser akzeptierte SQL-Anweisung muss darauf geprüft werden, ob der Anwender berechtigt ist, diese Anweisung für die angesprochenen Daten in der Datenbank überhaupt auszuführen. Diese Berechtigungen werden mithilfe spezieller SQL-Anweisungen eingerichtet (siehe Kapitel 12).

Update

Danach wird unterschieden, ob es sich um eine rein lesende Anfrage (Query) oder eine den Datenbankinhalt ändernde Abfrage handelt (siehe Kapitel 6 und 10). Bei ändernden Abfragen werden die im Data Dictionary hinterlegten Integritätsregeln geprüft. Dies kann beinhalten, dass ein Lagerbestand positiv bleiben muss, eine Bestellung einen Kunden haben muss oder ein Artikel sich einer bestehenden Warengruppe zuordnen lässt. Diese Regeln werden bei der Erstellung der Datenbank festgelegt. Damit wird hier verhindert, dass eine Änderung zu einer Verletzung dieser Regeln führt.

Query

Bei einer rein lesenden Abfrage wird geprüft, ob die Abfrage unnötig kompliziert gestellt ist und vielleicht vereinfacht werden kann. Außerdem muss in jedem Fall bei dem Zugriff des Benutzers auf ein externes Schema („VIEW") eine Umsetzung der verwendeten Begriffe auf das eigentliche Datenbankschema erfolgen. Diese Abfragen (SELECT) werden in Kapitel 4, 5, 9 und 11 ausführlich angesprochen.

Codegenerator

Die bis hierhin akzeptierte und eventuell umgeformte SQL-Anweisung wird dann vom Codegenerator übernommen, der die SQL-Anweisung in eine Folge von Lese- und Schreibbefehlen umsetzt, die in der Datenbank ausgeführt werden müssen, um die SQL-Anweisung inhaltlich richtig umzusetzen. Gleichzeitig wird hierbei durch die Wahl performanter Zugriffspfade und die Nutzung von Indizes und anderer physischer Zugriffs- und Speicheroptimierungen ein möglichst performanter und ressourcenschonender Code erzeugt. Auf diesen Prozess haben Sie mit SQL nur indirekt Einfluss, beispielsweise über die Einrichtung von Indizes (siehe Kapitel 14).

Transaktionsmanager

Problematisch ist, dass es sich nicht um einen, sondern um eine ganze Folge von Befehlen handelt. Da die Datenbank davon ausgehen muss, dass mehrere Benutzer gleichzeitig mit dem System arbeiten, muss sichergestellt werden, dass sich die Befehlsfolgen bei der Ausführung nicht gegenseitig beeinflussen. Daher steuert der Transaktionsmanager durch die Kapselung dieser Folgen zu Transaktionen eine sich gegenseitig möglichst wenig beeinträchtigende Abarbeitung dieser Befehlsfolgen, also letztlich der SQL-Anweisungen verschiedener Benutzer. Er liefert letztlich auch den Returncode und die Ergebnisse an den Ausgabeprozessor. Im Fehlerfall kann der Recoverymanager mithilfe des Logbuches den Zustand wiederherstellen, als wäre die SQL-Anweisung nie ausgeführt worden. Wie die Transaktionen gehandhabt werden sollen, kann in gewissem Umfang über SQL gesteuert werden (siehe Kapitel 13).

Physische Speicherverwaltung

Die unterste Ebene bildet die physische Speicherverwaltung, die die Schreib- und Lesebefehle umsetzt und an der zumeist mehrere Komponenten beteiligt sind, die eine möglichst optimale Nutzung des Puffers im Festspeicher (Hauptspeicher/Cache) sicherstellen sollen. Hier müssen möglichst viele benötigte Daten in schnellen Speichermedien zugreifbar sein. Die eigentliche physische Speicherung erfolgt zumeist seitenweise, also in größeren Blöcken, die vom Gerätemanager gelesen und geschrieben werden. Manche Datenbanken greifen hierzu ihrerseits auf Dateiverwaltungssysteme wie *InnoDB* oder andere Systeme zurück, die letztlich die physische Speicherverwaltung umsetzen. Diese können über eine eigene Transaktionssteuerung verfügen, sodass letztlich die Lese- und Schreibbefehle bereits für das Dateiverwaltungssystem erzeugt werden. Über SQL haben Sie hier nur über datenbankspezifische SQL-Erweiterungen und auch nur bedingt Einfluss.

2.4 Die Struktur von SQL-Anweisungen

SQL ist eine Programmiersprache und hat daher die üblichen syntaktischen Regeln, die ausführlich in den Standards und Handbüchern beschrieben werden. Mit vielen Teilaspekten wollen wir uns in den folgenden Kapiteln beschäftigen. Vorab an dieser Stelle aber ein paar wenige Grundregeln.

SQL-Anweisungen stehen immer allein für sich. Jede Anweisung bewirkt entweder die Bereitstellung einer Menge von Datensätzen aus der Datenbank oder die Änderung des Datenbankinhaltes.

Aufeinanderfolgende SQL-Anweisungen können normalerweise keine Informationen untereinander austauschen. Eine Folge von SQL-Anweisungen kann aber komplexe Datenbankänderungen realisieren, die dann weiteren Anweisungen zur Verfügung stehen.

Alle SQL-Anweisungen beginnen mit einem (englischen) Verb, zumeist gefolgt von dem Objekt, auf das sich die Anweisung bezieht, und näheren Spezifikationen. Die Tabelle 2.1 zeigt die Logik am Beispiel der SQL-DML, also des Teils von SQL, der der Abfrage und Änderung der Daten in der Datenbank dient.

Tabelle 2.1 Die Befehle der SQL-DML am Beispiel der Tabelle tbPerson

Befehl	Bedeutung
`SELECT * FROM tbPerson;`	*Wähle* alle Datensätze aus der Personentabelle, also lies alle Personeninformationen.
`UPDATE tbPerson SET ... ;`	*Aktualisiere* die Informationen in der Personentabelle, also die Informationen über eine oder mehrere Personen.
`DELETE FROM tbPerson ... ;`	*Lösche* einen oder mehrere Datensätze in der Personentabelle, lösche also bestimmte Personen aus dem Bestand.
`INSERT INTO tbPerson ...;`	*Einfügen* eines neuen Personendatensatzes, also die Aufnahme einer neuen Person in den Bestand.

Die Grundstruktur ist also einfach. Die SQL-Anweisung

```
SELECT * FROM tbPerson;
```

bewirkt ganz einfach, dass alle Informationen über Personen angezeigt werden. Das mögliche Ergebnis ist in Abbildung 2.3 wiedergegeben. Die Felder hängen natürlich von der Struktur der Tabelle ab. Typisch ist dabei die Darstellung in Tabellenform. Die Zeilen entsprechen jeweils einem einzelnen Datensatz, in diesem Fall allen verfügbaren Informationen über eine Person. Das gesamte Ergebnis ist eine Menge von Datensätzen. Die Reihenfolge ist dabei zunächst willkürlich. Die Spalten entsprechen den Datenfeldern in der Tabelle. Für jeden Datensatz existieren stets die gleichen Datenfelder.

Sollen nicht alle Informationen über eine Person ermittelt werden, sondern beispielsweise nur der Familienname, der Vorname und die Postleitzahl des Wohnortes, so lässt sich das Ergebnis auch sehr einfach auf die gewünschten Spalten beschränken. Zusätzlich können die Zeilen eingeschränkt werden, indem z.B. nur die Datensätze der Personen ermittelt werden, deren Postleitzahl 29221 ist. Es ergibt sich die SQL-Anweisung aus Listing 2.1.

Listing 2.1 Personen mit der Postleitzahl „29221“

```
SELECT Familienname, Vorname, Geburtsdatum
FROM tbPerson
WHERE PLZ = 29221;
```

PID	Familienname	Vorname	PLZ	Ort	Strasse	Geburtsdatum
1	Weiss	Peter	30529	Hannover	Palmstraße 6	07.11.1963
2	Bucz	Susanne	30531	Hannover	Heinestraße 23	06.04.1976
4	Karmann	Thomas	29227	Celle	Trift 28	04.08.1954
5	Klötzer	Karl	29221	Celle	Bahnhofstraße 2	13.03.1971
6	Weiss	Karin	30529	Hannover	Palmstraße 6	05.10.1962
7	Weiss	Peter	38134	Braunschweig	Glanweg 4	02.03.1974
8	Meier	Kathrin	38154	Braunschweig	Welfenallee 23	03.05.1981
9	Schmidt	Karl	30529	Hannover	Lavesallee 32	25.06.1949
10	Müller	Claudia	29596	Breitenhees	In den Fuhren 12	
11	Lisewski	Bernd	30890	Barsinghausen	Roggenkamp 10	06.06.1960
15	Martens	Melanie	29221	Celle	Horstweg 258	17.02.1961
17	Schlachter	Dieter	29227	Celle	Mondhagen 43	02.02.1961
23	Peredy	Helmut	29221	Celle	Mauernstraße 2	23.02.1956
24	Ruppert	Nicola	29301	Bergen	Welfenallee 23	25.02.1962
25	Sander		29223	Celle	Marxallee 12	05.02.1953
26	Cromberg	Jörg	38152	Braunschweig	Nordring 13	07.06.1991
31	Schulze	Tanja	29308	Winsen	Berliner Ring 23	09.11.1992
32	Winter	Petra	29320	Hermannsburg	Immenhoop 51	30.12.1989
34	Plate	Ulrich	30529	Hannover	Gutenberggasse 5	02.12.1986
37	Magerkurth	Melissa	29336	Nienhagen	Am Tümpel 3	04.09.1951

Abbildung 2.3
Alle Informationen über Personen

Wörtlich übersetzt heißt das:

```
LIES die Werte der FELDER Familienname, Vorname und Geburtsdatum
AUS der TABELLE tbPerson
aller Datensätze FÜR die gilt, dass die PLZ den Wert 29221 hat.
```

Listing 2.2
Deutsche Umschreibung des Listings 2.1

Die Grundstruktur der `SELECT`-Anweisung wird durch die sogenannten Schlüsselwörter (`KEYWORDS`) bestimmt, hier also:

SELECT ... FROM ... WHERE ...;

In dieses Gerüst sind anstelle der Punkte die benötigten Angaben einzufügen.

MS Access

Am Beispiel dieser SQL-Anweisung können Sie auch den Zusammenhang zwischen der eigentlichen SQL-Anweisung und der MS Access-Oberfläche erkennen. Abbildung 2.4 zeigt die Eingabe, die in MS Access für die Formulierung der obigen SQL-Anweisung notwendig ist. Im oberen Teil ist die Tabelle `tbPerson` dargestellt, die als Basis für die Informationen dienen wird. Der Name dieser „Datenquelle“ für die SQL-Anweisung wird in der SQL-Anweisung als

```
FROM tbPerson
```

wieder auftauchen. Die Felder der Tabelle werden im oberen Teil ebenfalls aufgelistet. Aus diesen verfügbaren Feldern werden die Felder der Abfrage ausgewählt. Diese Felder `Familienname`, `Vorname` und `PLZ` sind im unteren Teil des Fensters wiedergegeben. Aus ihnen erzeugt MS Access dann

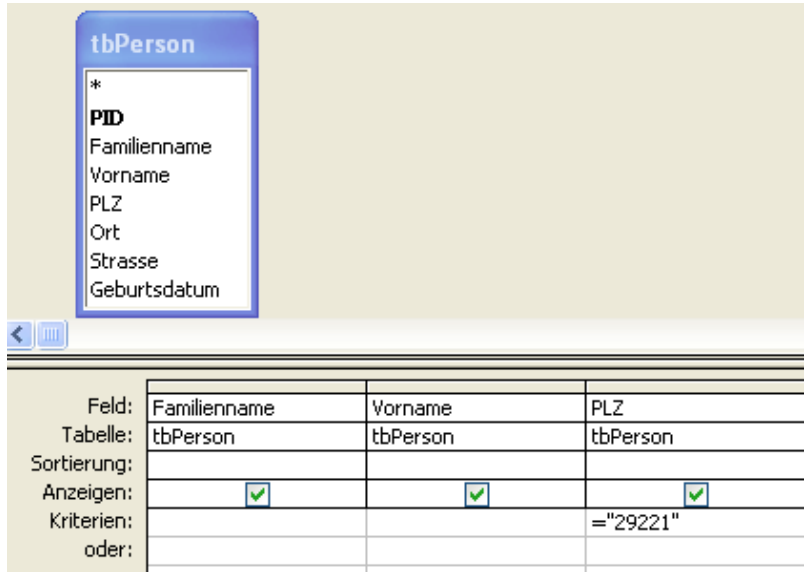
```
SELECT Familienname, Vorname, PLZ
```

als SQL-Anweisung. Man sieht der grafischen Darstellung also bereits die später enthaltenen Spalten an. Schließlich erfolgt noch die Auswahl der

Datensätze mit der Postleitzahl 29221. Diese Einschränkung wird als Eintrag in die Zeile KRITERIEN eingetragen. Hieraus entsteht dann in der SQL-Abfrage:

```
WHERE PLZ = 29221;
```

Abbildung 2.4
Auswahl dreier Felder aus der Tabelle tbPerson in MS-Access



Die Oberfläche beinhaltet weitere Eingabemöglichkeiten. So sehen Sie, dass auch eine Sortierung der Ergebnismenge vorgesehen ist. Zudem scheinen weitere Kriterien eingegeben werden zu können. Schließlich wird sich zeigen, dass noch weitere hier noch verborgene Eingabemöglichkeiten bestehen.

Ergebnis

Das Ergebnis der Beschreibung in Abbildung 2.4 führt zu der SQL-Anweisung aus Listing 2.1, das bei seiner Ausführung dann die Daten aus Abbildung 2.5 liefern könnte.

Abbildung 2.5
Ergebnisdaten der SQL-Anweisung

	Familienname	Vorname	PLZ
	Klötzer	Karl	29221
	Martens	Melanie	29221
	Peredy	Helmut	29221

SQL-Grundregeln

Somit haben wir jetzt eine erste SQL-Anweisung formuliert und auch den Bezug zu einer grafischen Oberfläche für deren Erstellung hergestellt. Es gibt einige Grundregeln für die Syntax von SQL-Anweisungen, die immer gelten:

- Eine SQL-Anweisung kann sich über mehrere Zeilen erstrecken, sie endet in der Regel mit einem Semikolon (;).

- SQL ist nicht „case-sensitiv“, Groß-/Kleinschreibung spielt also keine Rolle. Das gilt für die SQL-Schlüsselwörter wie `SELECT`, `FROM`, `INSERT` genauso wie für die Datenbankobjekte (Tabellen, Felder ...). Ausnahmen bilden hier Datenbanken wie openBase, die aber später noch zu besprechen sind.
- Bei den Datenbankinhalten, also den eigentlichen Daten, ist aber in den meisten Fällen auf Groß-/Kleinschreibung zu achten.

Info

SQL ist für die Verwaltung und Nutzung relationaler Datenbanken optimiert. Entsprechend ist SQL keine allgemeine Programmiersprache zur Lösung beliebiger Problemklassen. Wenn Sie Verzweigungen (IF, SWITCH, CASE, ...), Schleifen (WHILE, REPEAT, DO, ...), Unterprogramme und Prozeduren vermissen, so ist die Antwort: Das ist so (und lässt sich nur teilweise durch erweiterte Funktionen ändern). Das gibt es alles in SQL nicht. Benötigen Sie derartige Strukturen, müssen Sie die SQL-Anweisungen in andere Programmiersprachen einbetten. SQL konzentriert sich vollkommen auf die Objekte und Mengen relationaler Datenbanken.

Allerdings bieten die meisten Datenbankhersteller entweder eigene Erweiterungen zu SQL, wie Oracle mit PL/SQL, und/oder Einbettungen in andere Programmiersprachen, wie Visual Basic, Java oder andere Sprachen, an.

2.5 Relationale Datenbanken

SQL ist aufs Engste mit der Nutzung relationaler Datenbanken verbunden. Es wurde ausschließlich zu diesem Zweck entwickelt. Daher ist für das Verständnis von SQL zunächst ein zumindest oberflächliches Verständnis der Struktur relationaler Datenbanken notwendig. Dabei soll von der Technik abstrahiert werden und die Ebene des sogenannten Datenbankschemas gewählt werden, das mit SQL bearbeitet werden kann.

Das Datenbankschema nimmt eine Art Mittelposition zwischen der technischen Sicht und der Anwendersicht ein. So abstrahiert es einerseits von den technischen Details der eigentlichen Datenspeicherung und ist andererseits so formal, dass es eine Programmierung erlaubt. Das Datenbankschema einer relationalen Datenbank besteht im Wesentlichen zunächst aus Tabellen mit ihren Feldern (auch Spalten oder Attribute genannt) und (möglichen) Beziehungen zwischen diesen Tabellen.

2.5.1 Tabellen

Tabelle

Tabellen entstehen durch Gruppierung gleichartiger Daten. Typische Tabellen in der Datenbank eines Unternehmens sind „Artikel“, „Kunde“, „Auftrag“, „Rechnung“ ... In unserem Beispiel zur Kursverwaltung, das wir im nächsten Kapitel installieren wollen, finden Sie Tabellen wie `tbPerson`, `tbDozent` oder `tbKurs`.

Namen

Die Wahl von Namen für Tabellen wird von den meisten Unternehmen standardisiert. Um stets zu erkennen, um was es sich handelt – etwa um eine Tabelle, ein Feld oder andere Objekte wie einen VIEW oder eine Prozedur –, ist es sinnvoll, ein Präfix zur Kennzeichnung der verschiedenen Datenbankobjekte zu verwenden. Hier wird das Präfix „tb“ für Tabellen verwendet.

Besteht ein Name aus mehreren Begriffen, können diese mit Unterstrichen getrennt werden, also etwa `tbauftrag_position`. Ferner wird auch die sogenannte CamelCase-Schreibweise von Namen genutzt, bei der Namen, die logisch aus mehreren Wörtern bestehen, ohne Trennzeichen mit führendem Großbuchstaben direkt aneinandergesetzt werden, also etwa `tbAuftragPosition`. Dies hat den Vorteil, dass keine Konflikte mit den jeweiligen Namenskonventionen der verschiedenen Datenbankmanagementsysteme auftreten. Die Problematik der Groß-/Kleinschreibung muss in Abhängigkeit vom verwendeten Betriebssystem und Datenbanksystem berücksichtigt werden.

Jede Tabelle gruppiert die Informationen in Feldern. Eine Tabelle `tbArtikel` beschreibt beispielsweise, welche Informationen über einen Artikel gespeichert werden können. Für jede Information wird ein Feld verwendet. Ein Feld enthält **immer genau eine** Information – die Artikelnummer, die Artikelbezeichnung oder eine andere atomare Information –, aber nicht zwei oder mehrere Informationen gleichzeitig. Sind die Felder einer Tabelle `tbArtikel` bekannt, wissen wir, was wir an Informationen über einen Artikel speichern könnten.

Beispiel

Die Kursdatenbank, die hier als wesentliche Grundlage für die Beispiele verwendet werden wird, enthält unter anderem eine Tabelle `tbDozent`. Die Tabelle enthält alle grundlegenden Informationen über die Dozenten. Die Datenfelder sind in diesem Fall `DID`, `PID`, `Beschaeftigungsbeginn`, `Stundensatz`, `Firma`, `Titel` und `Qualifikationen`. Die Abbildung 2.6 zeigt einen Ausschnitt mit den ersten Datensätzen der Tabelle.

DID	PID	Beschaeftigungsbeginn	Stundensatz	Firma	Titel	Qualifikationen
812	1	01.07.2003	18,70 €	selbstständig	Informatiker	Word, Windows, Datenbanken
815	17	01.01.2002	15,00 €	selbstständig	IT-Kaufmann	Office, Datenbanken, Netzwerke
821	6	15.09.2005	13,00 €	Dreher KG	Meister	CNC-Steuerungen

Abbildung 2.6
Beispiel mit den ersten Datensätzen in einer Tabelle tbDozent

Attribut

Hier werden die einzelnen Informationen einer Tabelle als Felder bezeichnet. Dies ist eine weitverbreitete Bezeichnung. Synonym werden Felder auch als Spalten bezeichnet, da jedes Feld in der Darstellung einer Tabelle als Spalte erscheint. Auch der Begriff *Attribut* wird gelegentlich verwendet, wenn auch zumeist mehr im Zusammenhang mit der Modellierung einer Datenbank und weniger im Zusammenhang mit SQL.

Für jeden einzelnen Dozenten wird für jedes einzelne Feld festgelegt, welche Information tatsächlich gespeichert wird, also der konkrete Wert. So wird beispielsweise im Attribut `DID` (Dozenten-Identifikationsnummer) eine eindeutige Nummer für den Dozenten festgelegt, im Attribut `Stundensatz` finden Sie seinen Bruttoverdienst in Euro.

Datensatz (Tupel)

Jede Zeile einer Tabelle beschreibt einen Datensatz, hier einen Dozenten. Jeder Datensatz hat strukturell dieselben Felder, die aber natürlich für jeden Datensatz einen unterschiedlichen Wert haben können und im Normalfall auch haben. Datensätze beschreiben also jeweils einen kompletten Satz von Daten, der eine Person, einen Gegenstand, ein Konzept oder einen Prozess charakterisiert. Die Tabelle ist die Sammlung dieser gleich strukturierten Sätze. Aus dem mathematischen Konzept, das dem relationalen Modell zugrunde liegt, wird für den Datensatz auch der Begriff des Tupel abgeleitet und synonym zum Datensatz verwendet.

Datentyp

Jedes Feld einer Tabelle hat einen festgelegten Datentyp. Ein Datentyp für Texte ist beispielsweise `CHARACTER`, für Zahlen `INTEGER`, `FLOAT` oder `DECIMAL`. Daneben existieren spezielle Datentypen, wie `DATE`, `TIME` oder `TIMESTAMP`. Die Grundtypen sind in den einzelnen Datenbanksystemen weitgehend standardisiert, weitere Datentypen sind leider nicht einheitlich benannt und teilweise auch inhaltlich unterschiedlich realisiert, sodass sich hier in der Praxis immer wieder Probleme ergeben. Die grundsätzlichen Datentypen für Text, Zahlen und Datums-/Uhrzeitangaben sind aber in allen relationalen Datenbanken vorhanden und es muss auch in allen Systemen jedem Datenfeld ein Datentyp zugeordnet werden.

Sollten Sie sich wundern, warum der Dozent keinen Namen und keine Anschrift hat, so hat dies einen einfachen Grund. Diese Informationen sind in einer anderen Tabelle `tbPerson` enthalten, auf die hier nur über die Personen-Identifikationsnummer (`PID`) Bezug genommen wird. Diese Tabelle könnte die in Abbildung 2.7 dargestellten Datensätze enthalten.

Beziehung

Über einen gemeinsamen Wert in einem Feld – hier die PID – kann dann eine Beziehung hergestellt werden. So kann über den Wert „1“ als `PID` ermittelt werden, dass der Dozent 812, der Informatiker ist und dessen Qualifikationen Word, Windows und Datenbanken sind, Peter Weiss heißt und in Hannover in der Palmstraße 6 wohnt.

	PID	Familienname	Vorname	PLZ	Ort	Strasse	Geburtsdatum
	1	Weiss	Peter	30529	Hannover	Palmstraße 6	07.11.1963
	2	Bucz	Susanne	30531	Hannover	Heinestraße 23	06.04.1976
	4	Karmann	Thomas	29227	Celle	Trift 28	04.08.1954
	5	Klötzer	Karl	29221	Celle	Bahnhofstraße	13.03.1971
	6	Weiss	Karin	30529	Hannover	Palmstraße 6	05.10.1962
	7	Weiss	Peter	38134	Braunschweig	Glanweg 4	02.03.1974
	8	Meier	Kathrin	38154	Braunschweig	Welfenallee 23	03.05.1981

Abbildung 2.7
Ausschnitt aus der Tabelle tbPerson

2.5.2 Primärschlüssel

Jeder Datensatz einer Tabelle muss eindeutig identifizierbar sein. So muss beispielsweise jede Person in der Tabelle `tbPerson` eindeutig erkennbar sein. Da Namen und auch Vornamen mehrfach auftreten können, sind diese nicht eindeutig. Es wird ein zusätzliches Feld eingeführt, dessen Wert für jeden Datensatz in der Tabelle eindeutig ist: die Personen-Identifikationsnummer (`PID`). In Abbildung 2.7 ist der Feldwert „1“ des Feldes `PID` in der Tabelle `tbPerson` eindeutig. Mithilfe dieses Wertes kann der Datensatz des Dozenten Weiss eindeutig gefunden werden.

Wir haben uns in vielen Fällen im täglichen Leben bereits daran gewöhnt, dass alles eine Nummer bekommt: Personalausweise, Artikel, Rechnungen, Fahrgestelle, Flüge und Versicherungspolicen sind nur einige Beispiele.

Diese Nummern werden vergeben, weil die Unternehmen ihre Informationen in Datenbanken abspeichern. In diesen Datenbanken müssen die einzelnen Datensätze eindeutig identifizierbar und unterscheidbar sein. Dies geschieht mit sogenannten Schlüsseln oder genauer Primärschlüsseln (manchmal auch Identifizierungsschlüssel genannt).

Primärschlüssel

Ein Primärschlüssel (Primary key) ist ein Feld, mit dessen Wert jeder Datensatz in einer Tabelle eindeutig identifizierbar ist.

Für jeden Primärschlüssel in einer Datenbank muss stets gelten, dass ...

- er eindeutig jeden Datensatz in der Datenbank identifiziert.
- er sofort bei der Anlage eines Datensatzes vergeben wird.
- er sich während der Existenz des Datensatzes niemals ändern darf (oder bestimmte besondere Regeln beachtet werden müssen).

Sprechende Schlüssel

Primärschlüssel können für den Anwender erkennbare und interpretierbare Informationen enthalten, beispielsweise Anfangsbuchstaben von Kunden, Unterscheidungen von privaten und geschäftlichen Kunden, Geschäftsstellen und andere Angaben, die Mitarbeitern weiterhelfen. Man spricht in diesen Fällen von sprechenden Schlüsseln. Dies ist aber für die Datenbank nicht zwingend notwendig, da die Nummern im Wesentlichen der automatischen Suche in der Datenbank dienen. Oft existieren auch sprechende Schlüssel neben den in der Datenbank verwendeten Primärschlüsseln, die für die Anwender dann nie sichtbar werden.

Manchmal ergibt sich der für eine Tabelle sinnvolle Primärschlüssel auf natürliche Weise, beispielsweise die Fahrgestellnummer für die Identifizierung eines Autos, die EAN (Europäische Artikelnummer) als Primärschlüssel der Artikel aus dem Sortiment eines Handelsunternehmens oder die Bestimmung der Lage eines Ortes mithilfe seines Längen- und Breitengrades.

Künstliche Schlüssel

Oft sind diese natürlichen Schlüssel aber gar nicht vorhanden oder sie sind vorhanden, aber zu umständlich (Beispiel: Fahrgestellnummer), nicht eindeutig für das gesamte Sortiment eines Warenhauses (Beispiel: EAN nicht für alle Artikel vorhanden) oder sie sind einfach unhandlich. Wer weiß schon,

welche Längen- und Breitengradangaben der Palmstraße 6 in Hannover entsprechen? Und wenn Sie es wüssten, würden Sie es auf die fünfte oder sechste Dezimale genau eingeben wollen oder können? In diesen Fällen werden bei der Gestaltung der Datenbank künstliche Primärschlüssel eingeführt, also Felder, deren Werte nur der Identifizierung der Datensätze in einer Tabelle dienen. Sie besitzen in der realen Welt keine weiter gehende Bedeutung.

In unserer Kursdatenbank haben wir schon zwei solche Fälle gesehen. In Abbildung 2.7 sehen die `PID` (Personen-Identifikationsnummer), die in der Tabelle `tbPerson` dem eindeutigen Auffinden jedes Personendatensatzes dient. Die Werte haben keine interpretierbare Bedeutung. Entsprechendes gilt auch für die `DID` (Dozenten-Identifikationsnummer) in der Tabelle `tbDozent` (siehe Abbildung 2.6).

Warum nehmen wir nicht einfach den Namen einer Person als Primärschlüssel? Wir könnten Herrn Weiss doch auch eindeutig finden? Gut, ich habe oben schon behauptet, dass der nicht eindeutig ist, und Sie sehen es bereits in Abbildung 2.7. Wenn das aber nicht reicht, weil der Name „Weiss“ nicht eindeutig ist, könnte man doch den Vornamen dazunehmen. Aber was ist, wenn das auch nicht ausreichend ist?

Gut, auch das Problem sehen Sie in Abbildung 2.7. Wenn Sie die Datensätze mit den `PID` „1“ und „6“ betrachten, würde die Kombination der Datenfelder `Name` und `Vorname` gerade noch reichen, um jeden Datensatz eindeutig zu identifizieren. Spätestens beim Datensatz mit der `PID` „7“ reicht das aber auch nicht mehr aus. Jetzt müsste man ein weiteres Feld – beispielsweise die `Strasse` – dazunehmen. Aber reicht das wirklich in allen Fällen aus? Wer weiß, wer da nächsten Monat noch einzieht? Dieses Verfahren ist sehr unsicher. Aber nicht nur das, es ist auch umständlich und verkompliziert später die Realisierung der Beziehungen zwischen den Tabellen. Es ist daher naheliegend, einen künstlichen Primärschlüssel, in diesem Fall die `PID`, zu vergeben, der beim Aufbau der Datenbank mit Sicherheit eindeutig ist.

2.5.3 Beziehungen

Das grundlegende Strukturierungsmerkmal einer relationalen Datenbank sind also Tabellen für Personen, Dozenten, Kunden, Artikel, Bestellungen und andere. Nun wäre es sehr unbefriedigend, wenn man zwar feststellen könnte, welche Kunden man hat, aber nicht welcher Kunde welche Bestellung getätigt hat oder welcher Lieferant welche Artikel liefert oder welcher Dozent in welcher Straße wohnt. Daher dürfen die Tabellen nicht isoliert betrachtet werden, sondern müssen miteinander in Beziehung gebracht werden können. Erst diese Beziehungen ermöglichen übergreifende Auswertungen mit den entsprechenden Informationen.

Beziehung

Eine Beziehung (Relation) ist eine (mögliche) Kombination von Datensätzen aus zwei Tabellen. Ein Sonderfall sind rekursive Beziehungen, die eine Tabelle mit sich selbst in Beziehung setzen.

Bei einer Beziehung werden aus den beiden Tabellen diejenigen Datensätze als zusammenpassend angesehen, die in zwei bestimmten Feldern denselben

Wert haben. In einer der beiden Tabellen wird dazu zumeist der Primärschlüssel ausgewählt. In der zweiten Tabelle wird ein Feld bestimmt, dessen Werte mit den Werten des Primärschlüssels der anderen Tabelle übereinstimmen müssen. Dieses Feld wird als Fremdschlüssel bezeichnet.

Fremdschlüssel

Ein Fremdschlüssel ist ein Feld, dessen Werte mit den Werten des Primärschlüssels einer anderen Tabelle übereinstimmen. Dadurch werden diejenigen Datensätze beider Tabellen in Beziehung zueinander gesetzt, deren Werte im Primärschlüsselfeld und im Fremdschlüsselfeld übereinstimmen. Aus jeweils einem Datensatz beider Tabellen entsteht so virtuell ein gemeinsamer Datensatz mit allen Feldwerten aus beiden Tabellen. Abbildung 2.8 zeigt den Zusammenhang am Beispiel der beiden Tabellen `tbPerson` und `tbDozent`.

Fremdschlüssel
Tabelle tbDozent
DID | PID | Beschaeftigung | Stundensatz | Firma | Titel | Qualifikationen
812 | 1 | 01.07.2003 | 17,00 € | selbstständig | Informatiker | Word, Windows
815 | 17 | 01.01.2002 | 15,00 € | selbstständig | IT-Kaufmann | Office, Datenba
821 | 6 | 15.09.2005 | 13,00 € | Dreher KG | Meister | CNC-Steuerung
Primärschlüssel
Tabelle tbPerson
PID | Familienname | Vorname | PLZ | Ort | Strasse | Geburtsdatum
1 | Weiss | Peter | 30529 | Hannover | Palmstraße 6 | 07.11.1963
2 | Bucz | Susanne | 30531 | Hannover | Heinestraße 23 | 06.04.1976
4 | Karmann | Thomas | 29227 | Celle | Trift 28 | 04.08.1954
5 | Klötzer | Karl | 29221 | Celle | Bahnhofstraße 2 | 13.03.1971
6 | Weiss | Karin | 30529 | Hannover | Palmstraße 6 | 05.10.1962

Abbildung 2.8
Beziehung zwischen tbDozent und tbPerson über den Fremdschlüssel PID

Sie sehen, dass der Dozent mit der `DID` „812“ die `PID` „1“ hat. Da dies in der anderen Tabelle dem Datensatz mit dem Primärschlüssel `PID` mit dem Wert „1“ entspricht, ist klar, dass in Hannover in der Palmstraße 6 ein Peter Weiss wohnt, der als der entsprechende Dozent seit dem 01.07.2003 für einen Stundensatz von 17,- € unterrichtet, selbstständig und Informatiker ist.

Mehrere Primärschlüsselfelder

In der Praxis ergibt sich immer wieder die Problematik, dass Tabellen doch mehr als ein Primärschlüsselfeld aufweisen. In diesem Fall muss natürlich auch eine entsprechende Zahl an Fremdschlüsselfeldern vorhanden sein, die paarweise den Primärschlüsselfeldern entsprechen. Im Prinzip müssen zum Aufbau einer Fremdschlüsselbeziehung immer alle Primärschlüsselfelder der Tabelle, zu der die Beziehung aufgebaut werden soll, kopiert werden und eine entsprechende Anzahl passender Fremdschlüsselfelder angelegt werden.

Jetzt könnte hier noch ein SQL-Beispiel für die Abfrage über eine Beziehung hinweg stehen. Aber das würde dann doch zu viele Erklärungen erfordern. Wenn Sie wollen, können Sie aber schon einmal in Kapitel 4 unter dem Stichwort `JOIN` „spicken“, bevor wir uns jetzt den realen Datenbanken zuwenden.

3

Die Beispieldatenbanken

Bevor wir uns jetzt den ersten realen SQL-Befehlen im Detail zuwenden, wollen wir noch die beiden Grundlagen betrachten, die Sie für die Beispiele dieses Buches benötigen. Zum einen sind dies die Beispieldatenbanken, auf die sich die meisten SQL-Beispiele dieses Buches beziehen. Zum anderen soll die Installation der Datenbankmanagementsysteme MySQL, MS Access, openBase, Oracle und Firebird beschrieben werden, die Sie für die Bearbeitung der Beispiele nutzen können. Wir werden in diesem Buch Beispiele für alle fünf Systeme zeigen, indem wir auch für alle Übungsaufgaben Lösungen für alle Systeme erstellen, soweit dies sinnvoll möglich ist. Die Lösungen finden Sie auf der beiliegenden CD.

Dieses Kapitel hat zwei Funktionen. Es soll Ihnen zum einen helfen, eine Beispielumgebung für die Arbeit mit den Beispielen dieses Buches bereitzustellen. Es soll Ihnen zum anderen aber auch ein wenig Hintergrund zu den Beispielen und der Handhabung der Umgebungen liefern.

Für diejenigen, die sich nur für die Beispiele interessieren und die Datenbankinstallation bereits kennen, reicht es, die Abschnitte 3.1 und 3.2 zu lesen. Für die anderen Leser empfehlen sich auch die weiteren Kapitel, insbesondere diejenigen Abschnitte, die die Sie interessierenden Datenbanken beschreiben.

3.1 Die Kursdatenbank

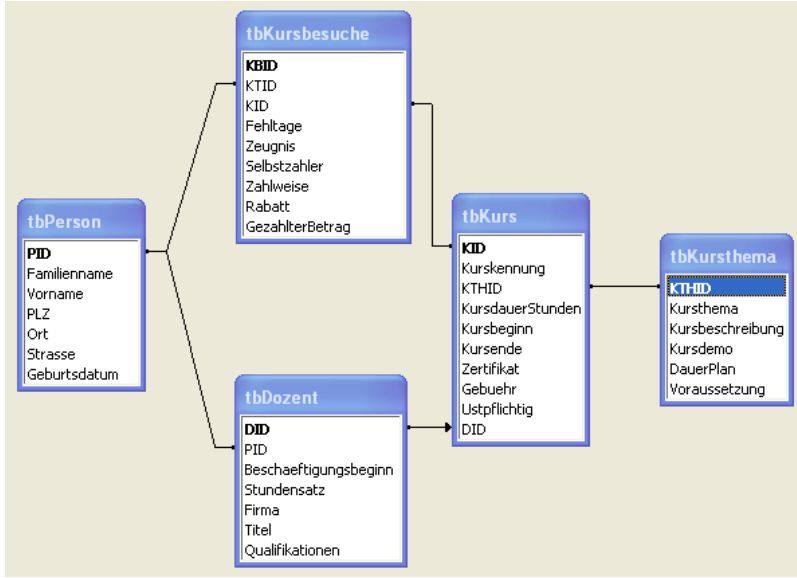
Kursdatenbank

Die wichtigste Beispieldatenbank ist die sogenannte „Kursdatenbank“. Sie beschreibt die Sicht eines Schulungsunternehmens auf seine Kurse. Dabei steht hier der Beispielgedanke im Vordergrund, viele Details sind daher nicht enthalten, die in der Praxis sicherlich eine zentrale Rolle spielen. Das Ziel der Datenbank soll aber auch nicht die reale Abbildung des Schulungsunternehmens sein, sondern ein möglichst verständliches Beispiel für die SQL-Anweisungen.

Datenmodell Kursdatenbank

Zunächst wollen wir uns daher das Modell mit seinen Tabellen und Feldern sowie den möglichen Beziehungen ansehen.

Abbildung 3.1
Datenbankschema der Kursdatenbank



Wie Sie in Abbildung 3.1 sehen, besteht die Datenbank aus fünf Tabellen. Kleine Ausschnitte der Daten in den Tabellen `tbPerson` und `tbDozent` haben wir in Kapitel 2 bereits betrachtet. Hier sehen Sie die kompletten Tabellen. Sie erkennen, dass im Datenbankschema die Datenfeldnamen vertikal in der Tabelle eingetragen sind. Werden die realen Daten gezeigt, wird die horizontale Darstellung verwendet, bei der die Datenfeldnamen die oberste Zeile bilden, da zumeist eine ganze Reihe von Datensätzen darunter angegeben werden sollen. Die beiden Tabellen sind mit einer Linie verbunden, die die Beziehung zwischen den Tabellen symbolisiert. Eine Linie verbindet beispielsweise das Primärschlüsselfeld `PID` der Tabelle `tbPerson` mit dem Fremdschlüsselfeld `PID` der Tabelle `tbDozent`. Dies bedeutet, dass inhaltlich die Dozenten mit einer bestimmten `PID` den Personen mit derselben `PID` entsprechen. Auf Details gehen wir später ein. Die Darstellung in Abbildung 3.1 ist MS Access entnommen, ähnliche Darstellungen werden aber von vielen Datenbanksystemen oder entsprechenden Hilfswerkzeugen angeboten.

tbPerson

Die Tabelle `tbPerson` beinhaltet die Stammdaten aller Personen, die mit dem Schulungsunternehmen zu tun haben, sei es als Teilnehmer oder Dozent. Die Verwaltungsangestellten und andere Mitarbeiter lassen sich hier natürlich ergänzen, sollen aber im Sinne einer besseren Übersichtlichkeit zunächst nicht betrachtet werden. Für jede Person sind außer ihrem Familiennamen und Vornamen noch die Postleitzahl, der Ort und die Straße sowie ihr Geburtsdatum hinterlegt.

tbDozent

Die Tabelle tbDozent beinhaltet zusätzliche Felder für alle Personen, die in dem Unternehmen als Dozenten tätig sind. So wird festgehalten, seit wann sie als Dozent beschäftigt sind, wie hoch ihr Stundensatz ist und welche Titel und Qualifikationen sie mitbringen. Da viele der Dozenten nebenberuflich tätig sind, wird außerdem der Name der Firma gespeichert, in der sie hauptberuflich beschäftigt sind. Über die durch eine Linie symbolisierte Beziehung zur Tabelle tbPerson kann ermittelt werden, welche Stammdaten, also Adresse und Geburtsdatum, die einzelnen Dozenten haben.

tbKursbesuche

Die Tabelle tbKursbesuche beinhaltet alle Teilnehmer an den Kursen. Sie ist wie die Dozententabelle über eine Beziehung mit der Tabelle tbPerson verbunden, in der sich die Basisdaten der Kursteilnehmer, insbesondere die Adressdaten befinden. In diesem Sinne bedeutet die Modellierung, dass es sich bei den Kursbesuchern immer um Personen handelt (tbPerson), die für einen Kurs die Rolle eines Kursteilnehmers (tbKursbesuche) annehmen können.

tbKursthema

Auf der anderen Seite der Grafik finden Sie die Tabelle tbKursthema, die alle angebotenen Kurse thematisch enthält, also jeweils das Thema des Kurses, eine ausführliche Beschreibung und weitere Angaben. Der Primärschlüssel dieser Tabelle ist die KTHID (KursTHema-Identifikationsnummer). Sie ist in Abbildung 3.1 beispielhaft hervorgehoben. Die Primärschlüssel der anderen Tabellen sind jeweils in Fettschrift markiert und nach demselben Schema benannt. Die Tabelle tbKursthema ist aber nicht im Sinne eines konkreten Kurses zu verstehen, sondern als Modul, das inhaltsgleich zu verschiedenen Terminen von verschiedenen Dozenten für verschiedene Teilnehmer angeboten werden kann.

tbKurs

Die konkreten Kurse sind in der Tabelle tbKurs aufgelistet. Hier werden die Termine angegeben sowie alle weiteren eventuell für die Durchführung relevanten Angaben gemacht. Über die Beziehung zur Tabelle tbKursthema kann auf die genauen Angaben zum Thema eines Kurses zurückgegriffen werden. Diese Angaben sind bei jeder Durchführung identisch, während sich die Termine, die konkrete Dauer, eine eventuelle Umsatzsteuerbefreiung und andere Angaben ändern können. Die Tabelle tbKurs beschreibt also die eigentliche Durchführung des Kurses, die darin besteht, dass zu dem geplanten Termin der Dozent und die Kursteilnehmer zusammengebracht werden (auf den Ort wurde hier noch verzichtet). Die Tabelle tbKurs ist somit das eigentliche Herzstück des Kursangebotes, obwohl sie vergleichsweise wenig Datenfelder enthält. Über die insgesamt drei Fremdschlüsselattribute werden das Kursthema, die Teilnehmer und der Dozent mit dem Kurs verbunden.

3.2 Schnelleinstieg und Neustart

Wenn Sie Ihre Datenbank bereits installiert haben, können Sie die Kursdatenbank und wenn Sie wollen auch das zweite Beispiel, die Artikeldatenbank, jetzt schnell installieren. Die Artikeldatenbank wird inhaltlich in den Kapiteln 7 und 8 ausführlich besprochen. Dieser Abschnitt richtet sich also

zunächst ausdrücklich nur an diejenigen Leser, die eine der hier verwendeten Datenbankumgebungen bereits lauffähig installiert haben und sich im Umgang damit bereits etwas auskennen. Für die anderen Leser erfolgt im Anschluss eine ausführliche Beschreibung.

Außerdem wird hier beschrieben, wie Sie bereits installierte Datenbanken wieder in den Ausgangszustand zurücksetzen können. Wenn Sie noch keine Datenbank installiert haben, überspringen Sie diesen Abschnitt und kommen darauf zurück, falls Sie später einmal Ihre Datenbank zurücksetzen möchten.

3.2.1 MySQL

kurse

Alle Daten für die Kursdatenbank stehen auf der CD unter *datenbanken/MySQL/kurse*. Um die Kursdatenbank wieder zurückzusetzen, wird das Skript *LoescheKursdb* aufgerufen und anschließend das Löschen mit „y“ bestätigt.

Um die Datenbank wieder zu laden, wird das Skript *Kursdb* aufgerufen.

Beide Skripte gehen davon aus, dass Sie das Passwort *masterkey* für den Benutzer *root* eingerichtet haben, wie es in der folgenden Installation empfohlen wird. Außerdem gehen beide Skripte davon aus, dass sie aus dem Verzeichnis *datenbanken/MySQL/kurse* heraus ausgeführt werden.

artikel

Alle Daten für die Artikeldatenbank stehen auf der CD unter *datenbanken/MySQL/artikel*. Um die Kursdatenbank wieder zurückzusetzen, wird das Skript *LoescheArtikeldb* aufgerufen und anschließend das Löschen mit „y“ bestätigt.

Um die Datenbank wieder zu laden, wird das Skript *Artikeldb* aufgerufen.

Beide Skripte gehen davon aus, dass Sie das Passwort *masterkey* für den Benutzer *root* eingerichtet haben, wie es in der folgenden Installation empfohlen wird. Außerdem gehen beide Skripte davon aus, dass sie aus dem Verzeichnis *datenbanken/MySQL/kurse* heraus ausgeführt werden.

3.2.2 Oracle

kurse

Alle Daten für die Kursdatenbank stehen auf der CD unter *datenbanken/Oracle/kurse*. Um die Kursdatenbank wieder zurückzusetzen, wird das Skript *LoescheOraKurse* aufgerufen.

Um die Datenbank wieder zu laden, wird das Skript *OraKurse* aufgerufen.

Beide Skripte gehen davon aus, dass Sie das Passwort *masterkey* für den Benutzer *system* eingerichtet haben, wie es in der folgenden Installation empfohlen wird. Die Existenz des Schemas können Sie in Oracle unter ADMINISTRATION/DATABASE USERS/MANAGE USERS kontrollieren. Existiert der Benutzer *kurse*, ist das Schema verfügbar. Sie können sich mit dem Benutzer *kurse* und dem Passwort *pwkurse* anmelden und unter OBJECT BROWSER/BROWSE/TABLES den Inhalt überprüfen. Im Tabellenblatt TABLE sehen Sie die Struktur, unter DATA den Inhalt der Tabellen.

artikel

Alle Daten für die Artikeldatenbank stehen auf der CD unter *datenbanken/Oracle/artikel*. Um die Kursdatenbank wieder zurückzusetzen, wird das Skript *LoescheOraArtikel* aufgerufen.

Um die Datenbank wieder zu laden, wird das Skript *OraArtikel* aufgerufen.

Beide Skripte gehen davon aus, dass Sie das Passwort *masterkey* für den Benutzer *system* eingerichtet haben, wie es in der folgenden Installation empfohlen wird. Die Existenz des Schemas können Sie in Oracle unter ADMINISTRATION/DATABASE USERS/MANAGE USERS kontrollieren. Existiert der Benutzer *artikel,* ist das Schema verfügbar. Sie können sich mit dem Benutzer *artikel* und dem Passwort *pwartikel* anmelden und unter OBJECT BROWSER/BROWSE/TABLES den Inhalt überprüfen. Im Tabellenblatt TABLE sehen Sie die Struktur, unter DATA den Inhalt der Tabellen.

3.2.3 Firebird

kurse

Alle Daten für die Kursdatenbank stehen auf der CD unter *datenbanken/Firebird/kurse*. Um die Kursdatenbank wieder zurückzusetzen, wird das Skript *LoescheFireKurse* aufgerufen.

Um die Datenbank wieder zu laden, wird das Skript *FireKurse* aufgerufen.

artikel

Alle Daten für die Artikeldatenbank stehen auf der CD unter *datenbanken/Firebird/artikel*. Um die Artikeldatenbank wieder zurückzusetzen, wird das Skript *LoescheFireArtikel* aufgerufen.

Um die Datenbank wieder zu laden, wird das Skript *FireArtikel* aufgerufen.

Beide Skripte (je Datenbank) gehen jeweils davon aus, dass Sie das Passwort *masterkey* für den Benutzer *SYSDBA* eingerichtet haben, wie es in der folgenden Installation empfohlen wird. Die Existenz des Schemas können Sie in Firebird mit der hier verwendeten Konsole unter DATABASES im Baum des linken Fensters prüfen. Mit einem Doppelklick auf die Datenbank öffnet sich eine Liste, die unter anderem den Eintrag TABLES zeigt. Wählen Sie diesen Eintrag aus, können Sie anschließend durch Doppelklick auf die gewünschte Tabelle im rechten Fenster ein weiteres Fenster mit der Struktur der Tabelle öffnen. In der Registerseite PROPERTIES sehen Sie die Struktur, unter DATA nach der zusätzlichen Auswahl von OPEN den Inhalt der Tabelle.

Wenn Sie eine Datenbank wie `kurse` oder `artikel` neu aufbauen wollen, können Sie dies durch den Start der beiden entsprechenden Skripte tun. Sie müssen sie hinterher unter Umständen noch registrieren. Dazu rufen Sie beispielsweise in der Konsolenoberfläche DATABASE/REGISTER auf. Lassen Sie sich auch nicht davon irritieren, dass die Datenbank nach der Ausführung des Löschskriptes noch sichtbar ist. Mit einem Doppelklick auf den Namen der Datenbank im linken Fenster erhalten Sie eine Fehlermeldung. Die Datenbank ist lediglich noch nicht abgemeldet, was auch nicht notwendig ist, da wir sie wieder neu erzeugen wollen.

3.2.4 MS Access

Alle Daten sind für MS Access 2003 vorbereitet. Um den Start zu vereinfachen, sind die Daten als komplette MS Access-Datenbank beigefügt und können unmittelbar genutzt werden.

kurse Alle Daten für die Kursdatenbank stehen auf der CD unter *datenbanken/MSAccess/kurse*. Die Datei *Kurse2003.mdb* enthält die komplette Kursdatenbank für MS Access 2003. Um die Kursdatenbank wieder zurückzusetzen, kopieren Sie die Datei einfach erneut von der CD. Für Nutzer älterer Versionen sind die Daten noch als Excel-Dateien beigefügt. Importieren Sie die Excel-Dateien in die Datenbank. Das Verfahren für MS Access wird in Anhang A beschrieben.

artikel Alle Daten für die Artikeldatenbank stehen auf der CD unter *datenbanken/MSAccess/artikel*. Die Datei *Artikel2003.mdb* enthält die komplette Artikeldatenbank für MS Access 2003. Um die Artikeldatenbank wieder zurückzusetzen, kopieren Sie die Datei einfach erneut von der CD. Für Nutzer älterer Versionen sind die Daten noch als Excel-Dateien beigefügt. Importieren Sie die Excel-Dateien in die Datenbank. Das Verfahren für MS Access wird in Anhang A beschrieben.

Denken Sie bei einer direkten Kopie von CD daran, gegebenenfalls den Schreibschutz zu entfernen. Klicken Sie dazu im Windows-Explorer mit der rechten Maustaste auf die kopierte Datei, wählen Sie den Befehl EIGENSCHAFTEN und entfernen Sie den Haken beim Schreibschutz.

3.2.5 openBase

Alle Daten sind für openOffice.org Base, hier immer kurz openBase genannt, vorbereitet. Um den Start zu vereinfachen, sind die Datenbanken jeweils als komplette Datenbanken beigefügt und können unmittelbar genutzt werden.

kurse Alle Daten für die Kursdatenbank stehen auf der CD unter *datenbanken/openBase/kurse*. Die Datei *kurse.odb* enthält die komplette Kursdatenbank für openBase. Um die Kursdatenbank wieder zurückzusetzen, kopieren Sie die Datei einfach erneut von der CD. Für Nutzer älterer Versionen sind die Daten noch als Calc-Dateien beigefügt. Importieren Sie die Calc-Dateien in die Datenbank. Das Verfahren für openBase wird in Anhang A beschrieben.

artikel Alle Daten für die Artikeldatenbank stehen auf der CD unter *datenbanken/openbase/artikel*. Die Datei *artikel.odb* enthält die komplette Artikeldatenbank für openBase. Um die Artikeldatenbank wieder zurückzusetzen, kopieren Sie die Datei einfach erneut von der CD. Für Nutzer älterer Versionen sind die Daten noch als Calc-Dateien beigefügt. Importieren Sie die Calc-Dateien in die Datenbank. Das Verfahren für openBase wird in Anhang A beschrieben.

Denken Sie bei einer direkten Kopie von CD daran, gegebenenfalls den Schreibschutz zu entfernen. Klicken Sie dazu im Windows-Explorer mit der rechten Maustaste auf die kopierte Datei, wählen Sie die EIGENSCHAFTEN und entfernen Sie den Haken beim Schreibschutz.

3.3 MySQL

3.3.1 Gründe für die Nutzung

MySQL ist ein Datenbankmanagementsystem, das von der MySQL AB entwickelt wurde und neben einer kommerziellen Variante auch unter der General Public License (GPL) verfügbar ist. Es darf damit für unsere Übungszwecke frei eingesetzt werden. Die breite Verfügbarkeit und die Lauffähigkeit unter Linux wie unter Windows haben es insbesondere für Internetserver, aber auch für viele mittelgroße Betriebe oder Abteilungen populär gemacht. Kurz vor der Fertigstellung dieses Buches wurde MySQL von Sun übernommen. Was dies für die Zukunft der Datenbank bedeutet, bleibt abzuwarten.

Hinzu kommt, dass MySQL sowohl über eine klassische zeilenorientierte SQL-Befehlseingabe als auch über zahlreiche grafische Front-Ends verfügt, mit denen komfortabel SQL-Befehle direkt eingegeben werden können. Man kann also sehr dicht am System sein, aber auch sehr komfortable Oberflächen nutzen. Für MySQL spricht also:

- Große Popularität und weite Verbreitung
- Freie Verfügbarkeit für die Leser
- Klassische SQL-Eingabemöglichkeiten verbunden mit einer grafischen Benutzeroberfläche

Die Beispiele beruhen auf der Version 5.0.41 für den Server und 5.0.11 für den Client. Natürlich gibt es laufend neuere Versionen, auf denen die Beispiele ebenfalls funktionieren sollten. Als Oberfläche wird MySQL Administrator 1.2.12 gewählt, eine ebenfalls weitverbreitete grafische Oberfläche.

3.3.2 Den Server installieren und konfigurieren

Sie finden die MySQL-Version auf der beiliegenden CD. Es werden hier Versionen des MySQL Community Servers verwendet. Neuere Versionen können Sie aus dem Internet herunterladen. Beachten Sie die jeweiligen Lizenzbedingungen, da sich diese für die verschiedenen Server ändern können.

Version

Zur Installation von MySQL verwenden Sie die Dateien im Unterordner *MySQL* auf der beiliegenden CD. Sie finden dort die gepackte Datei *mysql-5.0.41-win32.zip*. Für Linux sind unterschiedliche Versionen für die verschiedenen Systeme verfügbar, deren komplette Bereitstellung den Rahmen dieses Buches sprengen würde. Sie können sie für die Beispiele aber in gleicher Weise nutzen. Sie finden die Downloads unter `dev.mysql.com/downloads`.

Nach dem Entpacken und dem Start von *Setup.exe* führt Sie dann der Installationsassistent durch die weitere Installation.

Abbildung 3.2
Start des Setup-Wizards

MySQL Server 5.0 - Setup Wizard
Welcome to the Setup Wizard for MySQL Server 5.0
The Setup Wizard will install MySQL Server 5.0 release 5.0.41 on your computer. To continue, click Next.
WARNING: This program is protected by copyright law.
MySQL
< Back
Next >
Cancel

Nach dem Start meldet sich der Installationsassistent, wie in Abbildung 3.2 dargestellt. Die Installation kann in drei verschiedenen Varianten erfolgen (siehe Abbildung 3.3).

Abbildung 3.3
Verschiedene Installationsmöglichkeiten für MySQL

MySQL Server 5.0 - Setup Wizard
Setup Type
Choose the setup type that best suits your needs.
Please select a setup type.
Typical
Common program features will be installed. Recommended for general use.
Complete
All program features will be installed. (Requires the most disk space.)
Custom
Choose which program features you want installed and where they will be installed. Recommended for advanced users.
< Back
Next >
Cancel

Die Unterschiede zwischen den drei Varianten sehen Sie am besten, wenn Sie die individuelle Installation CUSTOM wählen und die komplette Übersicht über die verfügbaren Komponenten betrachten (siehe Abbildung 3.3). Wesentliche Unterschiede liegen insbesondere in der Installation von Entwicklungswerkzeugen, mit deren Hilfe man Anwendungen für MySQL schreiben kann.

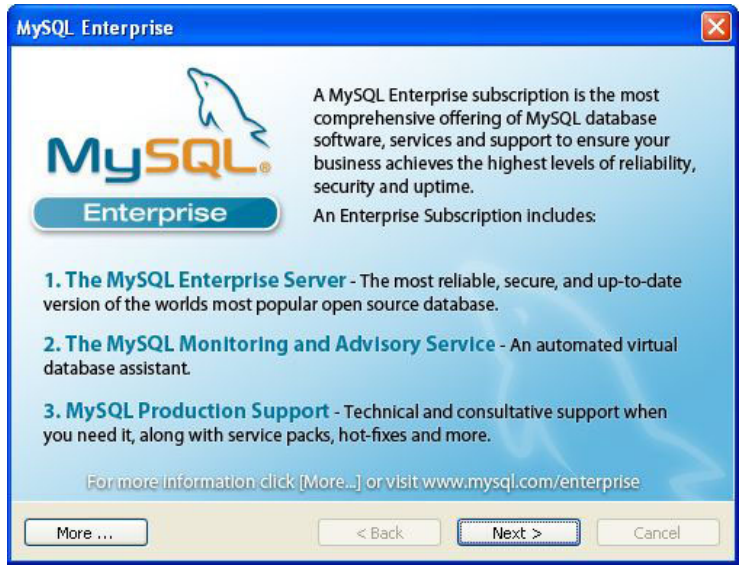
MySQL Server 5.0 - Setup Wizard
Custom Setup
Select the program features you want installed.
Click on an icon in the list below to change how a feature is installed.
MySQL Server
Client Programs
MySQL Instance Manager
Documentation
Developer Components
C Include Files / Lib Files
Scripts, Examples
Feature Description
Installs several helpful scripts and a couple of example files.
This feature requires 400KB on your hard drive.
Install to:
C:\Programme\MySQL\MySQL Server 5.0\
Change...
Help
< Back
Next >
Cancel

Abbildung 3.4
Benutzerspezifische Konfiguration

MySQL wird standardmäßig mit einer C-Bibliothek ausgeliefert, den *C Include Files/Lib Files*, mit denen SQL aus C-Programmen aufgerufen werden kann. Sie können die Entwicklungskomponenten zusätzlich auswählen, wenn Sie planen, mit MySQL auch Anwendungen zu entwickeln. Schnittstellen für andere Programmiersprachen existieren in großem Umfang, müssen aber getrennt installiert werden. Wollen Sie keine eigenen Programme schreiben, können Sie darauf verzichten und später bei Bedarf nachinstallieren.

In jedem Fall, egal, welche Konfigurationskomponente Sie gewählt haben, wird nach der Bestätigung die eigentliche Installation durchgeführt. Nach Abschluss der Installation und vor Beginn der Konfiguration erscheint der Hinweis auf die kommerzielle Version des Enterprise Servers sowie weitere Informationsquellen zu MySQL (siehe Abbildung 3.5).

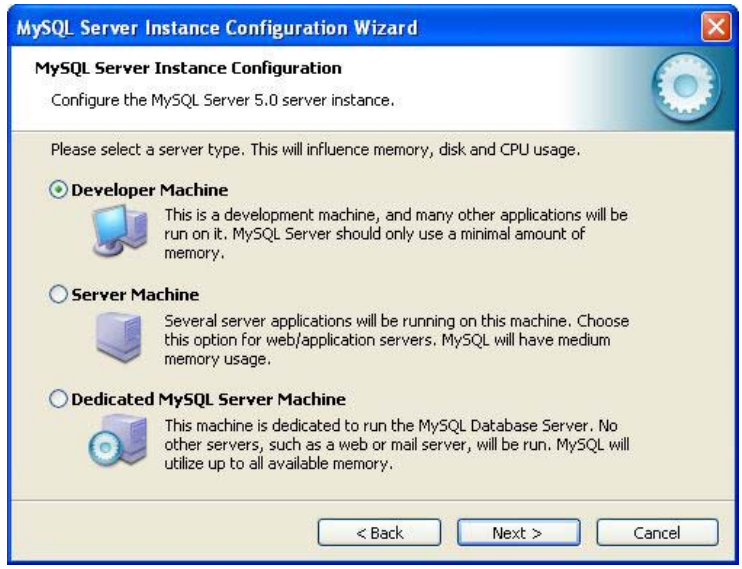
Abbildung 3.5
Bestätigung der Installation


MySQL Enterprise
MySQL
Enterprise
A MySQL Enterprise subscription is the most comprehensive offering of MySQL database software, services and support to ensure your business achieves the highest levels of reliability, security and uptime.
An Enterprise Subscription includes:
1. The MySQL Enterprise Server - The most reliable, secure, and up-to-date version of the worlds most popular open source database.
2. The MySQL Monitoring and Advisory Service - An automated virtual database assistant.
3. MySQL Production Support - Technical and consultative support when you need it, along with service packs, hot-fixes and more.
For more information click [More...] or visit www.mysql.com/enterprise
More ...
< Back
Next >
Cancel

Konfiguration

Nach einem weiteren Fenster beginnt die Konfiguration der Datenbank. Bei der Konfiguration haben Sie zunächst die Wahl zwischen der DETAILED CONFIGURATION und der STANDARD CONFIGURATION. Den allergrößten Teil der Beispiele dieses Buches können Sie mit der Standardkonfiguration bearbeiten. Gleichwohl gibt es gerade am Ende des Buches im Rahmen der technischen Beschreibungen einige Beispiele, für die bestimmte Datenhaltungssysteme notwendig sind. Das und das bessere Verständnis der Funktionsweise von MySQL sprechen daher für die Wahl der detaillierten Konfiguration, die dann einige weitere Angaben zu Ihrer Installation erfordert.

Abbildung 3.6
Basiskonfiguration


MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.0 server instance.
Please select a server type. This will influence memory, disk and CPU usage.
Developer Machine
This is a development machine, and many other applications will be run on it. MySQL Server should only use a minimal amount of memory.
Server Machine
Several server applications will be running on this machine. Choose this option for web/application servers. MySQL will have medium memory usage.
Dedicated MySQL Server Machine
This machine is dedicated to run the MySQL Database Server. No other servers, such as a web or mail server, will be run. MySQL will utilize up to all available memory.
< Back
Next >
Cancel

Developer Machine

Sie sehen drei typische Konfigurationen für die verschiedenen Nutzungsmöglichkeiten von MySQL. Die DEVELOPER MACHINE ist die typische Konfiguration für einen Entwicklungsrechner, auf dem die Datenbank einen Teil der Entwicklungsumgebung darstellt. Andere Entwicklungswerkzeuge und eventuell auch andere Datenbankmanagementsysteme werden parallel benötigt. Daher sollte MySQL nicht zu viel Speicher verwenden. Dazu kommt, dass in der Entwicklungsumgebung zumeist nicht allzu große Datenmengen verarbeitet werden müssen. Somit ist diese Konfiguration geeignet, um die Datenbank resourcenschonend zu betreiben, wenn nicht allzu hohe Anforderungen an die Performance gestellt werden.

Server Machine

Die SERVER MACHINE ist die typische Konfiguration für einen Webserver, bei dem MySQL schon größere Datenmengen und vor allem viele lesende Anfragen bearbeiten muss. Daher werden größere Ressourcen – insbesondere Speicher – auf dem Rechner benötigt. Andererseits müssen noch ein Webserver wie Apache und eine Reihe von PHP-, Python- oder Perlskripten für ein CMS, einen Webshop oder sonstige Anwendungen ausgeführt werden können, sodass nicht alle Ressourcen von MySQL beansprucht werden dürfen. Diese Konfiguration stellt also einen Kompromiss zwischen Performance der Datenbank und anderen Anwendungen dar.

Dedicated Server Machine

Schließlich gibt es noch den dedizierten Datenbankserver, sei es in einer Client/Server-Umgebung oder ebenfalls in einer Webumgebung mit verschiedenen Hardwareservern für die einzelnen Komponenten. In einer solchen Umgebung muss der Datenbankserver auf maximale Performance getrimmt werden und benötigt dafür alle verfügbaren Ressourcen.

Für unsere Zwecke sollte die Konfiguration als DEVELOPER MACHINE ausreichend sein.

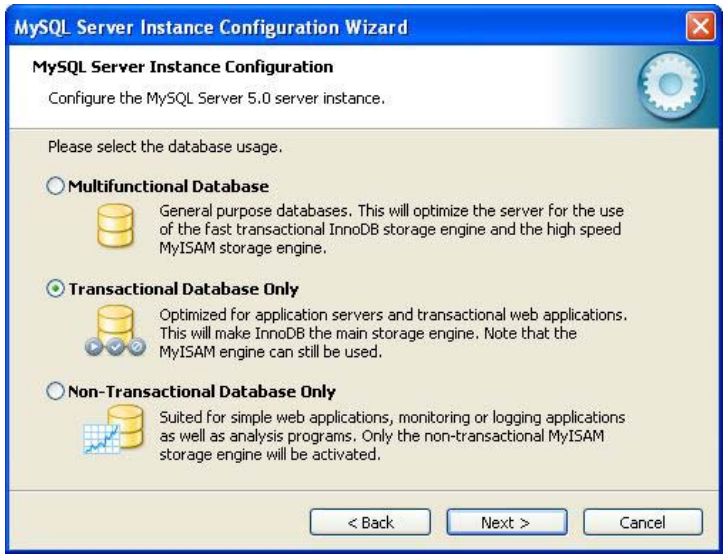

MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.0 server instance.
Please select the database usage.
Multifunctional Database
General purpose databases. This will optimize the server for the use of the fast transactional InnoDB storage engine and the high speed MyISAM storage engine.
Transactional Database Only
Optimized for application servers and transactional web applications. This will make InnoDB the main storage engine. Note that the MyISAM engine can still be used.
Non-Transactional Database Only
Suited for simple web applications, monitoring or logging applications as well as analysis programs. Only the non-transactional MyISAM storage engine will be activated.
< Back
Next >
Cancel

Abbildung 3.7
Nutzung verschiedener Datenablagesysteme

MySQL verwendet im Sinne einer Schichtenarchitektur verschiedene Datenhaltungssysteme, um die in der Datenbank abgelegten Informationen zu speichern. MySQL übernimmt also die Datenorganisation auf der Festplatte nicht selbst, sondern bedient sich dazu wiederum eines Datenhaltungssystems. Die beiden wichtigsten Datenhaltungssysteme sind *MyISAM* und *InnoDB*. Der wesentliche Unterschied ist, dass MyISAM zumeist schnellere Datenzugriffe bietet, dafür aber keine Transaktionen, also in sich abgeschlossene Prozessschritte, erlaubt. Solange Sie nur allein in der Datenbank arbeiten oder zumindest nur ein Benutzer ändernd auf die Datenbank zugreift, sind Transaktionen weitgehend überflüssig. Wenn Sie also MySQL nur zu Übungszwecken für dieses Buch einrichten wollen und die Transaktionsbeispiele in Kapitel 13 für Sie nicht relevant sind, reicht die Nutzung von MyISAM aus, im anderen Fall ist es besser, die Datenbank von vornherein für Transaktionen vorzubereiten. Wenn Sie TRANSACTIONAL DATABASE ONLY auswählen, stellt dies sicher, dass die Daten gleich in InnoDB verwaltet werden können. Für unsere relativ kleinen Datenmengen sollten in keinem Fall Probleme mit der Performance entstehen.

Alternativ können Sie die MULTIFUNCTIONAL DATABASE verwenden. Dann bleiben alle Optionen offen, Sie müssen dann aber die Verwendung später selbst steuern, was unter Umständen Komplikationen mit sich bringt.

Im Anschluss an die Wahl der „Multifunctional Database“ müssen Sie das Standardsystem für die physische Dateiverwaltung (InnoDB oder MyISAM) auswählen. Wenn Sie wirklich mit Transaktionen arbeiten wollen, sollten Sie InnoDB wählen. Sonst ist nur noch der Pfad für die Speicherung festzulegen.

Das nächste Auswahlfenster bestimmt die Anzahl gleichzeitiger Benutzer der Datenbank. Da wir in unserer Übungsdatenbank eher mit wenigen, maximal drei parallelen Nutzern arbeiten, können Sie entweder die Variante DECISION SUPPORT (DSS)/OLAP oder MANUAL SETTING mit einer Anzahl von beispielsweise fünf Nutzern wählen.

Im Folgenden geht es um die technische Basis des Servers. Unter Windows wird das System als Dienst eingerichtet. Für die Nutzung zusammen mit anderen Oberflächen ist es sinnvoll, wenn Sie TCP/IP aktivieren und den STRICT MODE ebenfalls einschalten (siehe Abbildung 3.8).

Danach müssen Sie den Standardzeichensatz festlegen, der in sogenannten „Collations“ gespeichert wird, siehe dazu auch Kapitel 8. Für unsere Zwecke reicht im Normalfall die Standardvariante Latin-1.

Danach aktivieren Sie MySQL endgültig als Windows-Service. Sie können den Standardnamen MySQL wählen. Starten Sie den Service automatisch mit Windows, wenn Sie ihn regelmäßig zum Testen und Probieren benötigen. Sonst sollten Sie ihn über den Windows-Dienstemanager jeweils aktivieren, wenn Sie ihn benötigen, da die Datenbank beim Start des Dienstes doch einige Zeit benötigt und Ressourcen belegt.

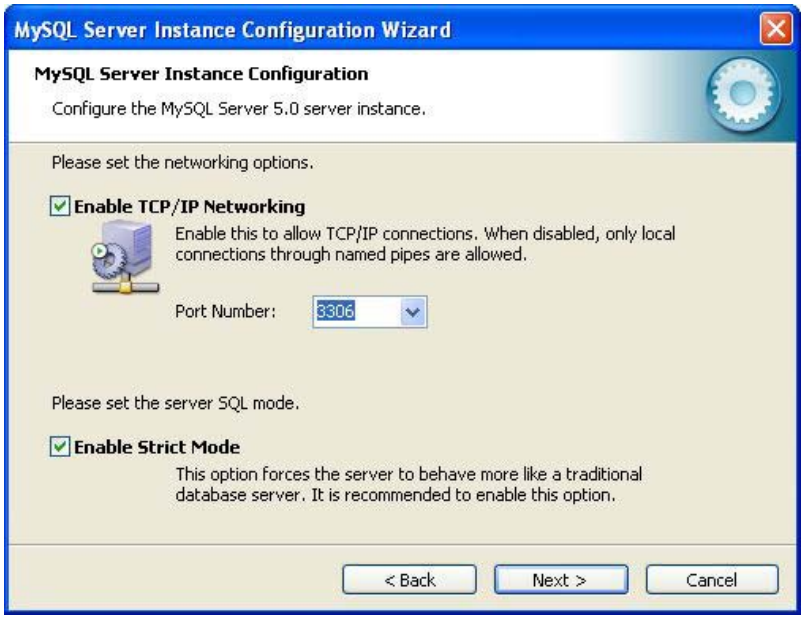

MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.0 server instance.
Please set the networking options.
Enable TCP/IP Networking
Enable this to allow TCP/IP connections. When disabled, only local connections through named pipes are allowed.
Port Number:
3306
Please set the server SQL mode.
Enable Strict Mode
This option forces the server to behave more like a traditional database server. It is recommended to enable this option.
< Back
Next >
Cancel

Abbildung 3.8
Technische Aktivierung

Die Option INCLUDE BIN DIRECTORY IN WINDOWS PATH ist dann von Vorteil, wenn Sie die Kommandozeilenwerkzeuge von MySQL nutzen wollen (Fenster EINGABEAUFFORDERUNG). In diesem Fall macht es Sinn, dass das BIN-Verzeichnis im Pfad liegt, weil Sie dann den Pfad beim Aufruf der einzelnen Kommandozeilenwerkzeuge nicht immer neu eingeben müssen.

Superuser = root

Nach der Definition des Servernamens und des Pfades sind alle wesentlichen Parameter für MySQL festgelegt. Damit Sie sich gleich erstmalig bei MySQL anmelden können, muss in der Datenbank noch mindestens ein Benutzerkonto eingerichtet werden, also ein Benutzer mit dessen Namen Sie sich bei MySQL identifizieren können. Wie jedes Datenbankmanagementsystem hat auch MySQL mindestens einen sogenannten Superuser. Er heißt hier *root*. Sie müssen jetzt das Passwort festlegen, mit dem sich *root* anmelden kann.

In diesem Buch wird als Passwort für den Superuser immer *masterkey* verwendet. Sie können natürlich ein anderes Passwort verwenden. Ersetzen Sie dann einfach an den entsprechenden Stellen der Listings und Skripte *masterkey* durch das entsprechende Passwort.

Tragen Sie ein Passwort ein und bestätigen Sie es (siehe Abbildung 3.9). Merken Sie sich das Passwort oder schreiben Sie es auf.

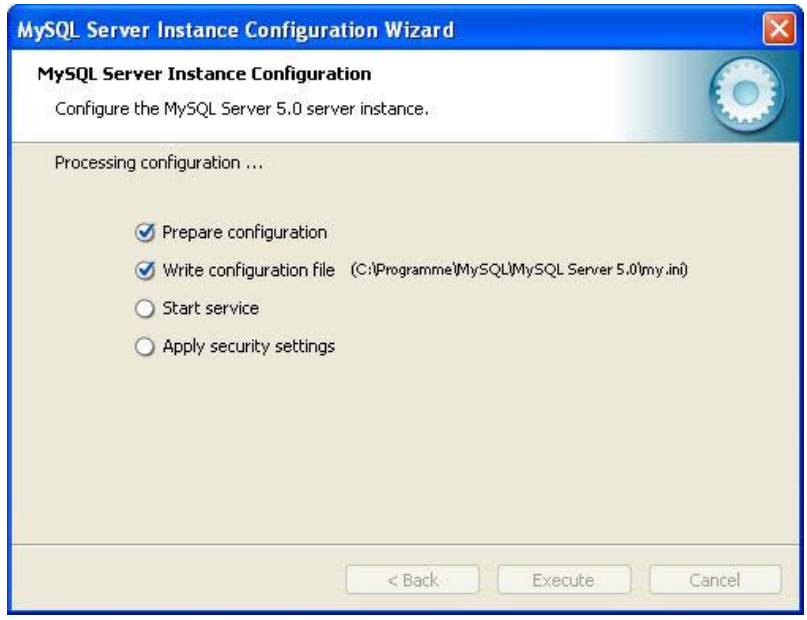
Den zusätzlichen anonymen Account benötigen wir für unsere Beispiele nicht zwingend.

Abbildung 3.9
Einrichten des Superusers in MySQL

MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.0 server instance.
Please set the security options.
Modify Security Settings
root
New root password:
*****
Enter the root password.
Confirm:
*****
Retype the password.
Enable root access from remote machines
Create An Anonymous Account
This option will create an anonymous account on this server. Please note that this can lead to an insecure system.
< Back
Next >
Cancel

Es beginnt die eigentliche Einrichtung des Servers, die Sie über eine Fortschrittsanzeige verfolgen können (siehe Abbildung 3.10).

Abbildung 3.10
Einrichtung und Start des Servers



Probleme gibt es unter Windows zumeist, wenn die Firewall eingeschaltet ist, weil der Port 3306, über den MySQL kommuniziert, nicht freigegeben ist. Sie erhalten dann eine Fehlermeldung wie in Abbildung 3.11.

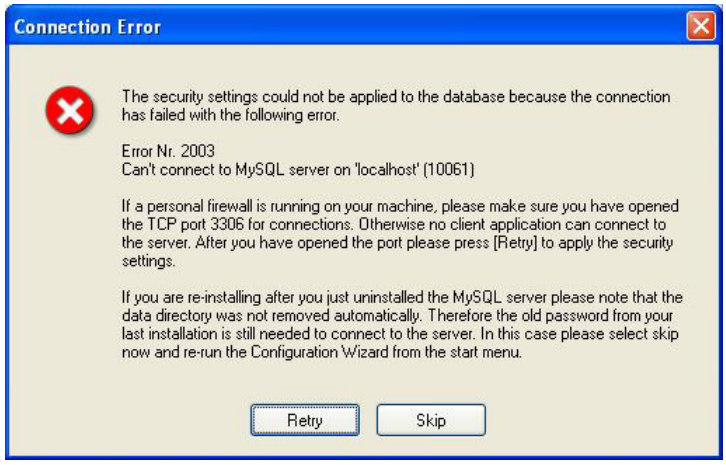

Connection Error
The security settings could not be applied to the database because the connection has failed with the following error.
Error Nr. 2003
Can't connect to MySQL server on 'localhost' (10061)
If a personal firewall is running on your machine, please make sure you have opened the TCP port 3306 for connections. Otherwise no client application can connect to the server. After you have opened the port please press [Retry] to apply the security settings.
If you are re-installing after you just uninstalled the MySQL server please note that the data directory was not removed automatically. Therefore the old password from your last installation is still needed to connect to the server. In this case please select skip now and re-run the Configuration Wizard from the start menu.
Retry
Skip

Abbildung 3.11
Fehlermeldung bei Konflikten mit der Portfreigabe in der Firewall

In diesem Fall müssen Sie den Port freigeben.

Windows-Firewall
Allgemein
Ausnahmen
Erweitert
Der Computer wird durch den Windows-Firewall geschützt.
Der Windows-Firewall trägt zum Schutz des Computers bei, indem der Zugriff durch nicht autorisierte Benutzer auf den Computer über das Internet bzw. ein Netzwerk verhindert wird.
Aktiv (empfohlen)
Diese Einstellung blockt den Zugriff auf den Computer für alle außen liegenden Quellen, mit Ausnahme derjenigen, die auf der Registerkarte "Ausnahmen" entsprechend ausgewählt sind.
Keine Ausnahmen zulassen
Aktivieren Sie dies, wenn Sie eine Verbindung mit einem öffentlichen Netzwerk von weniger sicheren Orten aus, wie z. B. Flughäfen, herstellen. Es wird keine Benachrichtigung angezeigt, wenn Programme geblockt werden. Die auf der Registerkarte "Ausnahmen" durchgeführte Auswahl wird ignoriert.
Inaktiv (nicht empfohlen)
Vermeiden Sie diese Einstellung. Durch Deaktivieren des Windows-Firewalls ist dieser Computer weniger vor Viren oder anderen Angriffen geschützt.
Weitere Informationen über den Windows-Firewall
OK
Abbrechen

Abbildung 3.12
Firewall unter Windows

Gehen Sie in die Systemsteuerung und öffnen Sie die Windows-Firewall. Unter Vista müssen Sie noch auf SICHERHEITSCENTER klicken, bevor Sie die Windows-Firewall öffnen können. Wenn Sie Ihre Firewall aktiviert haben (siehe Abbildung 3.12), können Sie diese natürlich deaktivieren. Ratsamer ist es aber, gezielt nur den von MySQL verwendeten Port freizugeben.

Dazu müssen zunächst Ausnahmen zugelassen sein. Gehen Sie auf AUSNAHMEN und wählen dort die Option PORT. Sie gelangen in ein kleines Fenster, in dem Sie den Port 3306 freigeben können.

Abbildung 3.13
Freigabe des TCP-Ports 3306 für MySQL

Port hinzufügen
Verwenden Sie diese Einstellungen, um einen Port durch den Windows-Firewall zu öffnen. Informationen betreffend der Suche der Portnummer und des Protokolls erhalten Sie in der Dokumentation des Programms bzw. des Diensts, den Sie verwenden möchten.
Name:
MySQL
Portnummer:
3306
TCP
UDP
Welche Risiken bestehen beim Öffnen eines Ports?
Bereich ändern...
OK
Abbrechen

Damit haben Sie alle Voraussetzungen geschaffen und die Installation von MySQL sollte normal enden (siehe Abbildung 3.14).

Der Server läuft jetzt auf dem System und kann genutzt werden. Sie können dies in der Windows-Systemsteuerung unter den Diensten kontrollieren.

Hier sollten Sie einen Dienst MYSQL finden, der gestartet ist. Je nachdem, ob Sie sich für oder gegen einen automatischen Start entschieden haben, finden Sie hier auch den entsprechenden Eintrag. Das Starten und Beenden des Dienstes können Sie ebenfalls manuell vornehmen.

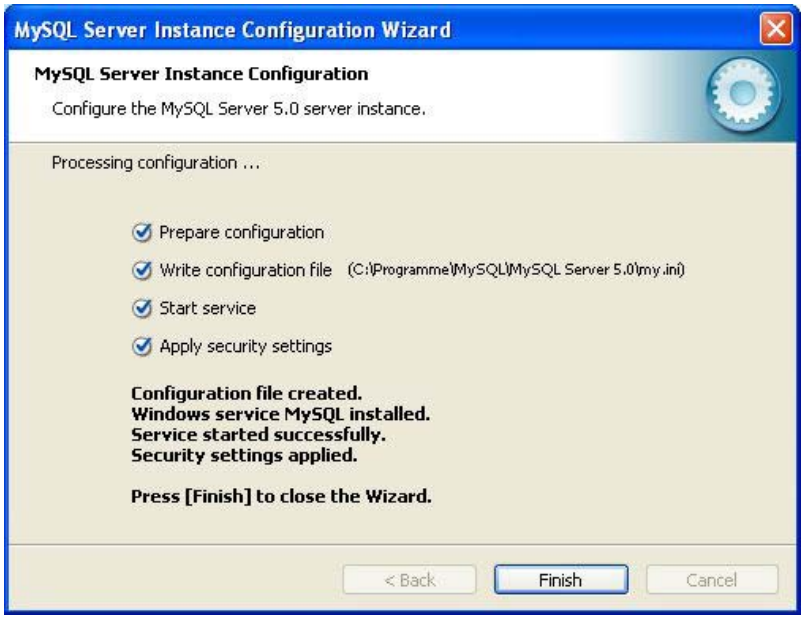


Abbildung 3.14
Abschluss der MySQL-Installation

3.3.3 Die Kommandozeile

MySQL bietet nach der Installation nur eine sehr eingeschränkte grafische Oberfläche. Hier macht die Installation einer Ergänzung wie der *MySQL Tools* Sinn.

Sie sollten aber auch die ganz elementare, einfache Kommandozeilenoberfläche von MySQL kennen, falls Ihnen einmal keine andere Oberfläche zur Verfügung steht oder falls Sie einmal Batchprozesse aufbauen und dabei die entsprechenden Komponenten nutzen wollen. Wir beschränken uns hier wieder auf die Windows-Sicht, unter Linux sieht die Umgebung aber prinzipiell sehr ähnlich aus.

Die Kommandooberfläche von MySQL wird direkt bei der Installation bereitgestellt. Installieren Sie MySQL unter Windows, wird das System direkt im Pfad *C:\Programme\MySQL\MySQL Server 5.0* abgelegt. Ein Unterverzeichnis, das dabei angelegt wird, ist das *\bin*-Verzeichnis. Einen Ausschnitt aus dem typischen Inhalt dieses Verzeichnisses sehen Sie in Abbildung 3.15.

Wir wollen zunächst nur mit den beiden Standardprogrammen *mysql.exe* und *mysqladmin.exe* arbeiten. Sie müssen dazu unter Windows über START/ALLE PROGRAMME/ZUBEHÖR das Zubehör von Windows wählen. Dort wiederum wählen Sie die EINGABEAUFFORDERUNG. Sie gelangen in das Fenster EINGABEAUFFORDERUNG. Hier müssen Sie in das MySQL-Verzeichnis navigieren (wenn Sie das Verzeichnis nicht bei der Installation in den Windows-Pfad aufgenommen haben). Falls Sie die alte DOS-Logik nicht mehr griffbereit haben, es reichen im Prinzip wenige Befehle, um dieses Verzeichnis zu erreichen.

Abbildung 3.15
Inhalt des Verzeichnisses …\mysql\bin (hier unter Windows)

Tabelle 3.1
Basisbefehle zum Erreichen der Kommandozeile von MySQL

Befehl	Bedeutung
`cd verzeichnis`	Wechsle in das angegebene Unterverzeichnis
`cd \`	Wechsle auf das Basislaufwerk, aus jedem beliebigen Unterverzeichnis möglich
`laufwerk:`	Wechsle in das angegebene Laufwerk
`cd …`	Wechsle in das übergeordnete Verzeichnis
`dir`	Zeige alle Dateien im aktuellen Verzeichnis an

Beispiel

Nach dem Aufruf des Fensters für die Eingabeaufforderung befinden Sie sich beispielsweise an der unten angegebenen Stelle, während MySQL unter *C:\Programme\MySQL\MySQL Server 5.0\bin* installiert ist.

```
d:\Dokumente und Einstellungen\Harry>
```

Sie geben ein:

```
C:
cd Programme
```

```
cd MySQL
cd MySQL Server 5.0
cd bin
dir
```

Jetzt müssten Sie in etwa das Bild aus Abbildung 3.15 sehen.

mysqladmin.exe

Das Programm *mysqladmin.exe* dient der Administration von MySQL, erfüllt also etwa die Funktion des MySQL Administrators. Hiermit können die grundlegenden Anweisungen für den technischen Betrieb des Datenbankmanagementsystems erteilt werden. Mit dem Aufruf

```
C:\Programme\mysql\MySQL Server 5.0\bin>mysqladmin -help
```

können Sie die Optionen aufrufen, die beim Aufruf des Programms angegeben werden können. Der Pfad muss nur eingegeben werden, wenn Sie sich nicht in dem Verzeichnis befinden bzw. der Pfad nicht bei der Installation eingetragen wurde. Im Folgenden werden wir dies voraussetzen und ihn weglassen.

So können Sie sich beispielsweise als Benutzer *root* anmelden und prüfen, ob der MySQL-Server antwortet:

```
mysqladmin -uroot -pmasterkey ping
```

Beachten Sie, dass nach dem Parameter, beispielsweise nach `-p`, der Wert – hier das Passwort – direkt und ohne Leerzeichen angeschlossen wird. Sie können auch mit

```
mysqladmin -uroot -pmasterkey version
```

die aktuelle Version Ihres Servers und einige weitere Statusinformationen erhalten. Wir werden in Abschnitt 3.3.5 *mysqladmin.exe* nutzen, um ein Datenbankschema für unsere Übungsdatenbanken anzulegen.

mysql.exe

Das zweite wichtige Programm ist *mysql.exe*. Es dient der Eingabe beliebiger SQL-Anweisungen. Während also *mysqladmin.exe* die grundlegenden technischen Einstellungen steuert, können mit *mysql.exe* die für uns interessanten SQL-Befehle eingegeben werden. Der Aufruf, um die verfügbaren Parameter zu erhalten, lautet:

```
mysql -?
```

Denken Sie gegebenenfalls an den Pfadnamen. Sie sehen, dass Sie eine Reihe von Optionen, wiederum beginnend mit einem Bindestrich, angeben können. Die einfachste Form der Anmeldung bei der Datenbank ist:

```
mysql -uroot -pmasterkey
```

Nach diesem Befehl ändert sich der Cursor zu:

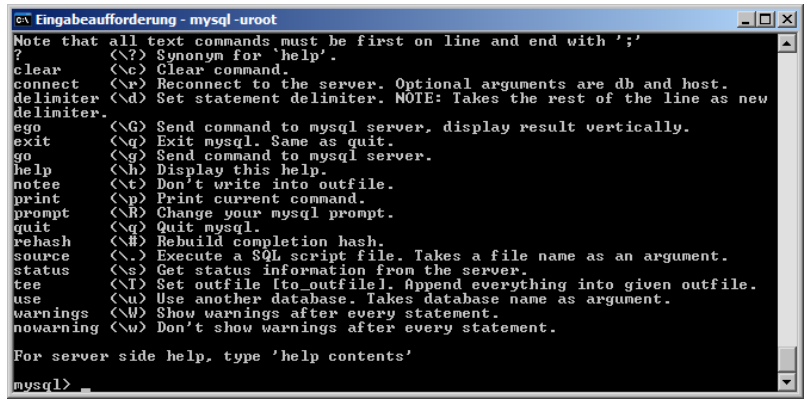
```
mysql>
```

Diese unscheinbare Änderung besagt, dass Sie jetzt nicht mehr auf Betriebssystemebene sind, sondern sich innerhalb eines MySQL-Clients bewegen. Sie sind beim Datenbankmanagementsystem angemeldet und können ab jetzt direkt SQL-Anweisungen sowie einige zusätzliche Kommandos eingeben. Testen Sie dies, indem Sie beispielsweise

```
help
```

eingeben. Sie erhalten eine Liste der möglichen Kommandos, die etwa der Liste in Abbildung 3.16 entsprechen sollte. Die meisten dieser Optionen spielen für uns hier keine große Rolle. Wir wollen aber im nächsten Abschnitt die Funktionsweise am Beispiel der ersten Beispieldatenbank zeigen.

Abbildung 3.16
Optionen von mysql.exe


Eingabeaufforderung - mysql -uroot
Note that all text commands must be first on line and end with ';'
? (\?) Synonym for `help'.
clear (\c) Clear command.
connect (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter. NOTE: Takes the rest of the line as new delimiter.
ego (\G) Send command to mysql server, display result vertically.
exit (\q) Exit mysql. Same as quit.
go (\g) Send command to mysql server.
help (\h) Display this help.
notee (\t) Don't write into outfile.
print (\p) Print current command.
prompt (\R) Change your mysql prompt.
quit (\q) Quit mysql.
rehash (\#) Rebuild completion hash.
source (\.) Execute a SQL script file. Takes a file name as an argument.
status (\s) Get status information from the server.
tee (\T) Set outfile [to_outfile]. Append everything into given outfile.
use (\u) Use another database. Takes database name as argument.
warnings (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
For server side help, type 'help contents'
mysql>

Verlassen Sie dazu zunächst *mysql.exe* mit dem Kommando

`exit` (oder `quit`).

Sie gelangen auf die Betriebssystemebene zurück, wie Sie an dem veränderten Cursor sehen.

3.3.4 Die grafische Oberfläche MySQL Tools

MySQL meldet sich nach dem Start ausschließlich als Dienst. Dieser besitzt auf Windows eine kleine Oberfläche, die aber nicht zur Eingabe von SQL-Anweisungen geeignet ist. Die zeichenorientierte Oberfläche, die unmittelbar zur Verfügung steht, haben wir gerade getestet. Sie ist letztlich wenig komfortabel. Daher sollten Sie zusätzlich eine grafische Oberfläche installieren. Eine der verfügbaren Oberflächen ist ebenfalls im Unterverzeichnis *MySQL* der CD vorhanden, die *MySQL Tools*. Sie können sie auch direkt von MySQL und verschiedenen Websites laden.

Zur Installation der MySQL Tools starten Sie *mysql-gui-tools 5.0-r12-win32.msi* aus demselben Verzeichnis.

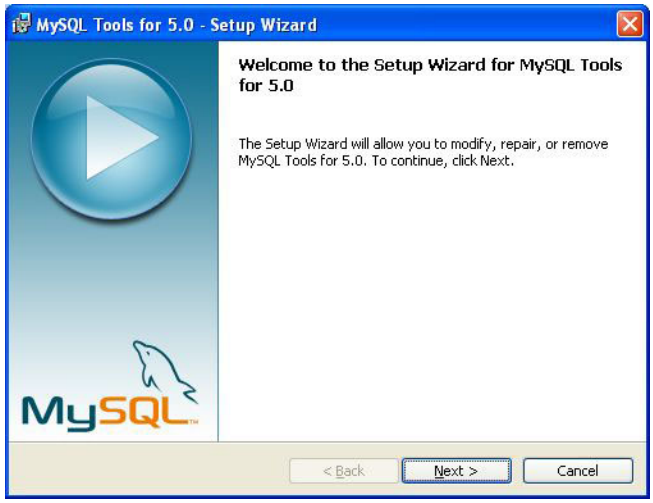

MySQL Tools for 5.0 - Setup Wizard
Welcome to the Setup Wizard for MySQL Tools for 5.0
The Setup Wizard will allow you to modify, repair, or remove MySQL Tools for 5.0. To continue, click Next.
MySQL
< Back
Next >
Cancel

Abbildung 3.17
Start der Installation von MySQL Tools

Nach dem Start können Sie die Installation neu erstellen beziehungsweise modifizieren. Eine benutzerdefinierte Installation (CUSTOM SETUP) zeigt Ihnen die einzelnen Komponenten:

- Der SQL-Administrator dient der Verwaltung der Datenbank, also des Service, der Benutzer, der Verbindungen, der Datensicherung und anderer Aktivitäten, die nichts mit dem eigentlichen Inhalt in Form der Tabellen zu tun haben.
- Der Query Browser ist das eigentliche Werkzeug für die Eingabe und Ausführung der SQL-Anweisungen. Die meisten Anweisungen unserer Beispiele sind hier einzugeben.
- Das Migration Toolkit dient der Datenmigration und wird uns nicht weiter beschäftigen.

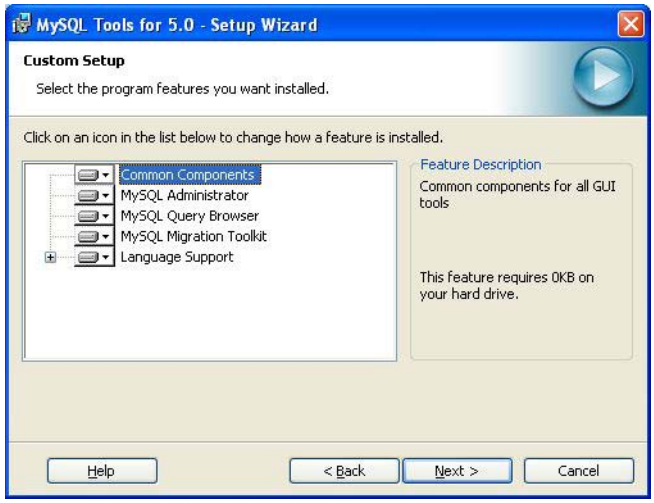

MySQL Tools for 5.0 - Setup Wizard
Custom Setup
Select the program features you want installed.
Click on an icon in the list below to change how a feature is installed.
Common Components
MySQL Administrator
MySQL Query Browser
MySQL Migration Toolkit
Language Support
Feature Description
Common components for all GUI tools
This feature requires 0KB on your hard drive.
Help
< Back
Next >
Cancel

Abbildung 3.18
Die verschiedenen Komponenten der grafischen Tools

Nach der Bestätigung erfolgt die Installation. Sie können nach der Installation die Programme über das normale Start-Menü von Windows aufrufen. Starten Sie den MySQL-Administrator für Ihre erste Testanmeldung. Sie werden aufgefordert, sich zu identifizieren (siehe Abbildung 3.19).

Abbildung 3.19
Anmeldung als root

Sie müssen die Adresse des Rechners mit dem Datenbankserver angeben. Dies kann eine TCP/IP-Adresse oder der Name des Rechners im Netzwerk sein. Arbeiten Sie lokal auf einer Maschine, können Sie entsprechend *127.0.0.1* oder einfach *localhost* angeben. Sie erinnern sich, dass MySQL den Benutzer *root* bei der Installation eingerichtet hat? Geben Sie daher hier als Benutzername *root* ein und das Passwort, das Sie bei der Installation gewählt haben, normalerweise *masterkey*. Mehr Informationen haben wir noch nicht und benötigen sie im Moment auch noch nicht.

Nach der erfolgreichen Anmeldung gelangen Sie in das Übersichtsfenster mit allen relevanten Informationen. Die verschiedenen Optionen auf der linken Seite werden uns noch in dem einen oder anderen Fall beschäftigen.

Über den Menüpunkt EXTRAS gelangen Sie zu den anderen Werkzeugen, die auch direkt aufgerufen werden können.

Sie können beispielsweise auch Ihre Verbindung zur Datenbank speichern, wenn Sie sie für künftige Anmeldungen verwenden wollen (siehe Abbildung 3.21).

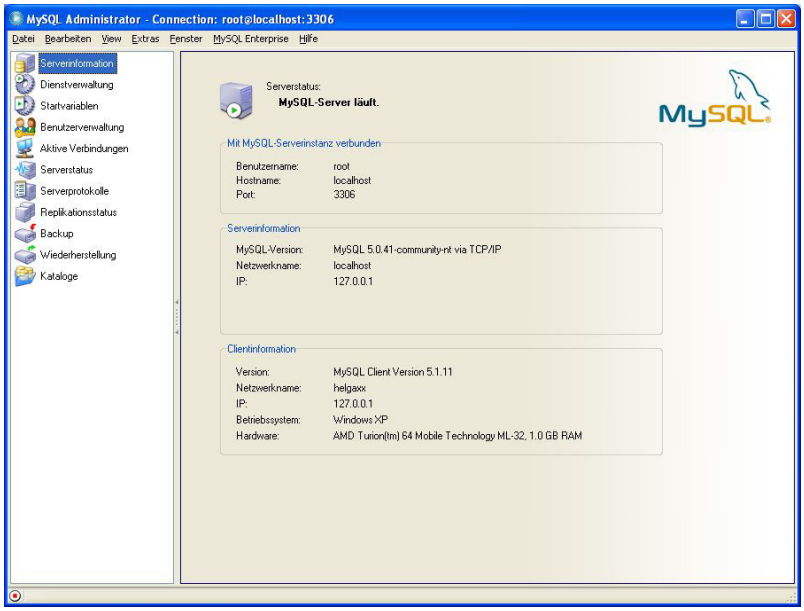


Abbildung 3.20
Übersichtsfenster des MySQL Administrators

Abbildung 3.21
Menüpunkt Extras

Damit ist die Installation der Arbeitsumgebung für MySQL abgeschlossen.

Nach der Installation können Sie die Oberfläche direkt als Windows-Programm starten. Es gibt im Wesentlichen vier Programme unter START/ALLE PROGRAMME/MYSQL:

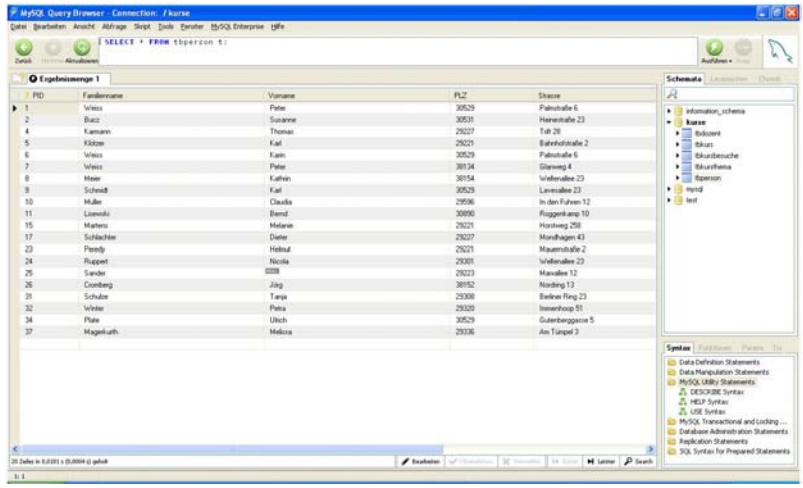
- MySQL Administrator
- MySQL Migration Toolkit
- MySQL Query Browser
- MySQL System Tray Monitor

Der Monitor dient der Überwachung des Systems. Mit ihm kann MySQL auch gestartet werden. Sollten Sie beim Aufruf des Administrators oder des Query Browsers eine Fehlermeldung erhalten, so starten Sie mit dem Monitor mit START INSTANCE eine neue Instanz. Sie finden den Monitor unten in der Taskleiste. Sie öffnen ihn mit der rechten Maustaste, die linke startet und beendet den Server ebenfalls.

Der Administrator dient der Administration des Systems, also der Verwaltung von Benutzern und Speicherplatz, der Datensicherung und Wiederherstellung von Daten. Wir wollen darauf in Kapitel 12 zurückkommen.

Der MySQL Query Browser dient der Ausführung der SQL-Kommandos. Er entspricht der *mysql.exe*. Nach dem Start und der Anmeldung erhalten Sie eine Ansicht wie in Abbildung 3.22.

Abbildung 3.22
Oberfläche MySQL Query Browser



Die Grundstruktur erlaubt im oberen Fenster die Eingabe von SQL-Anweisungen. Jede SQL-Anweisung, die Sie im oberen Fenster eingeben, muss durch Drücken der Schaltfläche AUSFÜHREN im rechten Bereich ausgeführt werden. Das Ergebnis wird im unteren Hauptfenster – dem Ergebnisbereich – angezeigt. Dieser Bereich verfügt über Tabs, die immer einschließlich der SQL-Anweisung gelten, sodass Sie immer mit einer Vielzahl von Anweisungen parallel arbeiten können. Rechts haben Sie die verfügbaren Datenbankschemata, die sich erweitern lassen, sodass die Tabellen und auch deren Felder sichtbar werden. Unten finden Sie noch eine sinnvolle Hilfe zur SQL-Syntax.

Weitere Hilfen zum Umgang finden Sie in Anhang A.

3.3.5 Eine Beispieldatenbank aufbauen

Um mit MySQL eine Datenbank aufzubauen, sind wie bei fast allen Datenbankmanagementsystemen drei Schritte notwendig. Das entspricht auch der Logik von SQL.

1. Erstellen des Datenbankschemas (manchmal spricht man auch vereinfacht von der Datenbank)
2. Erstellen der Tabellen (auf die weiteren Strukturen verzichten wir hier noch)
3. Eingabe der eigentlichen Daten

Schritt 1: Datenbankschema erstellen

Das Datenbankschema ist der „Container“, in dem eine Gruppe zusammengehöriger Tabellen zusammengefasst wird. Dies ist die gängige Gruppierung für Projekte, Anwendungen und andere in sich mehr oder weniger geschlossene Informationsbereiche.

Wir wollen jetzt die Datenbankschemata für die Übungsdatenbanken anlegen. Dazu verwenden wir den Benutzer *root*, der bei der Installation eingerichtet wird.

Geben Sie zunächst im Fenster mit der Eingabeaufforderung folgenden Befehl ein, um ein Datenbankschema `kurse` einzurichten:

```
mysqladmin.exe -uroot -pmasterkey CREATE kurse
```

Die wesentliche Syntax ist dabei wie folgt:

Aufruf mysqladmin.exe

```
<pfadname>mysqladmin.exe
        -uBenutzer [-pPasswort] [CREATE|DROP] Datenbankschema
```

Mit `CREATE` wird das Datenbankschema erzeugt, wobei es noch nicht vorhanden sein darf. In diesem Fall wird die Anweisung zum Schutz der in dem Schema enthaltenen Daten abgewiesen. Mit `DROP` können Sie ein Schema einschließlich aller enthaltenen Daten löschen.

Die Abkürzung `u` steht für User, `p` für Password. Beachten Sie, dass kein Leerzeichen oder irgendein anderes Zeichen zwischen dem `-u` und dem eigentlichen Benutzernamen respektive zwischen dem `-p` und dem Passwort stehen darf.

Alternativ können Sie als Syntax `--user=root` und `--password=Passwort` verwenden:

```
mysqladmin.exe --user=root --password=masterkey CREATE kurse
```

Schritt 2: Tabellen erstellen

Aufruf mysql.exe

Nachdem das Datenbankschema eingerichtet ist, kann jetzt auf das Tool *mysql.exe* zurückgegriffen werden, das für normale Anwender ausreichend ist. Die Syntax lautet hier:

```
<pfadname>mysql.exe
        -uBenutzer [-pPasswort] [Datenbankschema] [< Inputskript]
```

Bearbeitung der Skripte

Das Inputskript enthält wie alle hier verwendeten Skripte Toolaufrufe und SQL-Anweisungen, die ausgeführt werden sollen. Sie können die Skripte mit einem normalen Editor wie dem *Notepad* bearbeiten. Seien Sie aber vorsichtig mit Textverarbeitungsprogrammen wie Word, da Skripte keine Formatierungsanweisungen enthalten dürfen. Prüfen Sie die Skripte und passen Sie Pfade oder Passwörter gegebenenfalls an Ihre Umgebung an.

Tabellen anlegen

Jetzt kann es losgehen. Zunächst werden die Tabellen angelegt. Dazu können Sie das vorbereitete Skript *CreateKurse.txt* nutzen. Alle Dateien, die jetzt angesprochen werden, finden Sie auf der CD im Verzeichnis *datenbanken\MySQL\kurse*.

Die Datei *CreateKurse.txt* beginnt mit folgendem Inhalt:

Listing 3.1
CREATE TABLE tbPerson

```
DROP TABLE IF EXISTS tbPerson;

CREATE TABLE IF NOT EXISTS tbPerson (
PID int NOT NULL PRIMARY KEY,
Familienname varchar(50) NOT NULL,
Vorname varchar(50) NULL,
PLZ char(5) NULL,
Strasse varchar(50) NULL,
Ort varchar(50) NULL DEFAULT 'Celle',
Geburtsdatum date);
```

Im Prinzip handelt es sich dabei um zwei SQL-Anweisungen. Zunächst wird geprüft, ob die Tabelle schon existiert, und diese gegebenenfalls gelöscht. Dann wird eine neue Tabelle `tbPerson` erzeugt. Für diese Befehle benötigen wir den Administrator nicht, da es hier „nur“ um Inhalte geht. Diese SQL-Befehle können wir mit *mysql.exe* ausführen lassen. Wechseln Sie in der Eingabeaufforderung in das Kurs-Verzeichnis */datenbanken/MySQL/kurse* und schicken Sie dann folgenden Befehl ab:

```
mysql.exe -uroot –pmasterkey kurse < CreateKurse.txt
```

Diese Zeile bedeutet im Prinzip:

1. Melde dich als Benutzer root bei MySQL an. (Gedanklich haben wir jetzt den Cursor `mysql>` vor uns.)
2. Wähle die Datenbank kurse (entspricht einem `mysql>use Kurse`).
3. Führe alle SQL-Befehle aus, die du in der Datei *CreateKurse.txt* findest.
4. Führe die Abmeldung aus (entspricht `mysql>exit`).

Zusätzlich zum Benutzer und zum Passwort muss jetzt das Datenbankschema angegeben werden, das wir im ersten Schritt bereits erzeugt haben. Damit weiß MySQL, in welchem Datenbankschema die Tabellen angelegt werden sollen. Schließlich wird das Skript *CreateKurse.txt* als Input-Datenstrom an das Tool übergeben. Die Verzeichnisstruktur entspricht der Struktur auf Ihrer CD. Bei einer abweichenden Installation müssen hier eventuell Änderungen vorgenommen werden.

Ergebniskontrolle I

Wir wollen das Ergebnis unserer „Batch“-SQL-Anweisungen zunächst mit *mysql.exe* überprüfen. Dazu wollen wir mit direkten Befehlen und SQL-Anweisungen in *mysql.exe* arbeiten. Zunächst müssen wir uns also wieder anmelden:

```
mysql –uroot -pmasterkey
```

Nach diesem Befehl ändert sich der Cursor wie gehabt zu

```
mysql>
```

Jetzt müssen wir das richtige Datenbankschema wählen, dazu gibt es die `use`-Option (siehe Abbildung 3.16).

```
mysql>use kurse;
```

Gedanklich befinden wir uns jetzt innerhalb von MySQL und hier wiederum innerhalb der Datenbank `kurse`. Damit haben wir auf alle Tabellen dieser Datenbank Zugriff. Wir wollen dies beispielhaft überprüfen, indem wir uns den Aufbau der Tabelle `tbPerson` anzeigen lassen:

```
mysql>describe tbPerson;
```

Das Ergebnis sollte bei Ihnen etwa wie in Abbildung 3.23 aussehen. Vergleichen Sie das Ergebnis einmal in Ruhe mit der Anweisung `CREATE TABLE tbPerson`, die wir oben als Auszug der Datei *CreateKurse.txt* in Listing 3.1 gesehen haben. Sie sehen das unmittelbare Ergebnis dieser SQL-Anweisung.

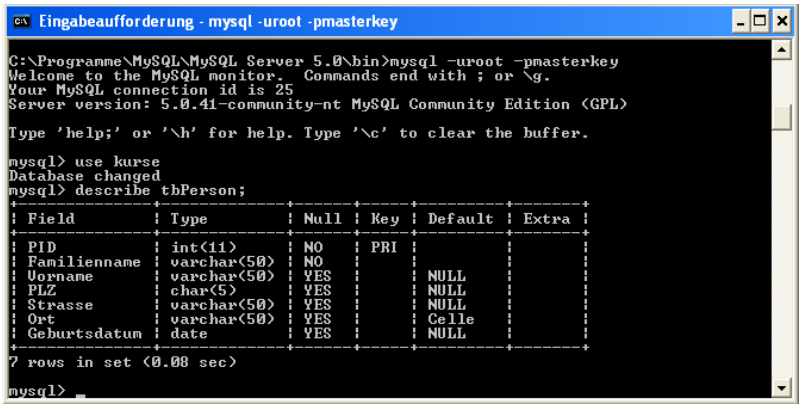

Eingabeaufforderung - mysql -uroot -pmasterkey
C:\Programme\MySQL\MySQL Server 5.0\bin>mysql -uroot -pmasterkey
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 25
Server version: 5.0.41-community-nt MySQL Community Edition (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use kurse
Database changed
mysql> describe tbPerson;
+---------------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
PID	int(11)	NO	PRI		
Familienname	varchar(50)	NO			
Vorname	varchar(50)	YES		NULL	
PLZ	char(5)	YES		NULL	
Strasse	varchar(50)	YES		NULL	
Ort	varchar(50)	YES		Celle	
Geburtsdatum	date	YES		NULL	
+---------------+-------------+------+-----+---------+-------+
7 rows in set (0.08 sec)
mysql>

Abbildung 3.23
Tabelle tbPerson im Kommandozeilenmodus

Ergebniskontrolle II

Wenn Sie das kleine Beispiel mit der zeilenorientierten Oberfläche durchgearbeitet haben, finden Sie jetzt das Datenbankschema `kurse`, das bereits fünf Tabellen enthält, auch in der Oberfläche der MySQL-Tools. Rufen Sie das Werkzeug MySQL Query Browser auf (direkt oder über EXTRAS in MySQL Administrator). Sie finden im rechten Teil des Bildschirms die Schemata. Eventuell müssen Sie diese aktualisieren (rechte Maustaste), siehe Abbildung 3.24. Mit der rechten Maustaste können Sie außerdem ein Schema zum aktuellen Schema bestimmen. Dieses wird dann in Fettschrift dargestellt. Nur die Tabellen des aktuellen Schemas können direkt verwendet werden, alle anderen Tabellen müssen mit dem Schemanamen qualifiziert werden, also `kurse.tbPerson`, statt einfach `tbperson`. Sie sollten daher das Schema `kurse` als aktuelles Schema auswählen.

Die SQL-Anweisung

```
DESCRIBE tbPerson;
```

können Sie jetzt direkt im oberen Fenster eingeben und ausführen (AUSFÜHREN). Sie sollten dann den Inhalt aus Abbildung 3.25 erhalten. Dies ist eine komplette Beschreibung der Tabellenstruktur. Sie sehen in jeder Zeile die Beschreibung eines Datenfeldes mit Namen, Datentyp, der Angabe, ob das Feld leer bleiben darf (NULL), ob es Primärschlüssel ist und ob ein Standardwert vorhanden ist. Die Details werden wir in Kapitel 8 betrachten.

Abbildung 3.24
Das Fenster Schemata in MySQL Query Browser mit dem Schema kurse

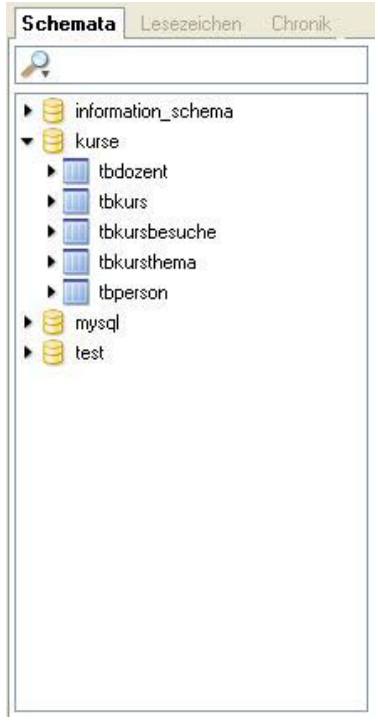


Abbildung 3.25
Struktur der Tabelle tbPerson

Ergebnismenge 1

Field	Type	Null	Key	Default	Extra
KID	char(10)	NO	PRI		
Kurskennung	char(50)	NO			
KTHID	int(11)	NO			
KursdauerStunden	int(11)	YES		NULL	
Kursbeginn	date	YES		NULL	
Kursende	date	YES		NULL	
Zertifikat	char(50)	YES		NULL	
Gebuehr	float	YES		NULL	
Ustpflichtig	char(1)	YES		NULL	
DID	int(11)	YES		NULL	

Sie haben vielleicht im unteren rechten Teil bereits ein kleines Fenster entdeckt. Dort erhalten Sie Hilfe zu verschiedenen SQL-Aspekten. Diese Hilfe ist sinnvoll, wenn Sie die grundsätzlichen Befehle und die Struktur von SQL verstanden haben. Sie können dann leicht die diversen spezifischen Parameter und Optionen einer SQL-Anweisung nachschlagen. Öffnen Sie beispielsweise MySQL Utility Statements, so finden Sie hier neben dem bereits bekannten `USE` zur Auswahl eines Datenbankschemas auch die SQL-Anweisung `DESCRIBE`. In der Hilfe zu `DESCRIBE` erfahren Sie, dass Sie auch die Abkürzung `DESC` verwenden sowie gezielt einzelne Spalten (columns) auswählen können, beispielsweise mit:

```
DESC tbPerson Ort;
```

Schritt 3: Daten importieren

Nach der Erzeugung der Tabellenstruktur werden jetzt in einem dritten Schritt die eigentlichen Daten importiert. Dafür wird wieder *mysql.exe* mit identischer Syntax verwendet.

Um gleich eine weitere Tabelle kennenzulernen, sollen zunächst Daten in die Tabelle `tbKurs` geladen werden. Prüfen Sie zunächst die Struktur der Tabelle mit:

```
DESC tbKurs;
```

Wenn Sie auf Ihrer CD in die Datei *Kurse.txt* schauen (*\datenbanken\MySQL\kurse*), sehen Sie folgende Zeilen:

Listing 3.2
Einfügen von Daten in die Tabelle tbKurs

```
INSERT INTO tbKurs (KID,Kurskennung,KTHID,KursdauerStunden,Kursbe-
ginn,Kursende,Zertifikat,Gebuehr,Ustpflichtig,DID)

VALUES('CE17','Celle17-Word',4,40,'2007-04-23','2007-04-
27','eigen',280.00,'J',833);

INSERT INTO tbKurs (KID,Kurskennung,KTHID,KursdauerStunden,Kursbe-
ginn,Kursende,Zertifikat,Gebuehr,Ustpflichtig,DID)

VALUES('CE20','Celle20-Word',4,40,'2007-07-09','2007-07-
13','eigen',280.00,'J',812);

INSERT INTO tbKurs (KID,Kurskennung,KTHID,KursdauerStunden,Kursbe-
ginn,Kursende,Zertifikat,Gebuehr,Ustpflichtig,DID)

VALUES('CE23','Celle23-Access',5,36,'2007-08-06','2007-08-
10','eigen',350.00,'N',812);
...
```

Es folgen noch eine Reihe weiterer Zeilen. Bei der Datei handelt es sich wieder um SQL-Anweisungen, mit denen die Daten in die Tabelle `tbKurs` eingefügt werden können. Dies erreichen wir mit der Zeile:

```
mysql.exe -uroot –pmasterkey kurse < Kurse.txt
```

Die angegebenen Daten werden in die Tabelle `tbKurs` geladen.

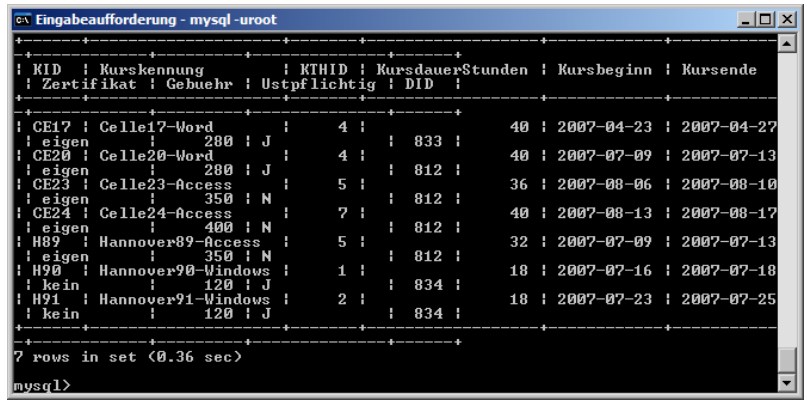
Ergebniskontrolle III

Wir wollen das Laden der Daten in die Tabelle `tbKurs` überprüfen. Dazu genügt uns eine andere einfache SQL-Anweisung, vielleicht die SQL-Anweisung schlechthin:

```
mysql>SELECT * FROM tbKurs;
```

Mit dieser Anweisung werden alle Daten angezeigt, die in der Tabelle `tbKurs` gespeichert sind. Das Ergebnis können Sie der Abbildung 3.26 entnehmen. Achten Sie dabei auf den Zeilenumbruch. Jeder Datensatz und auch alle Erläuterungszeilen sind aus Platzgründen in zwei Zeilen umgebrochen. Vergleichen Sie die ersten drei Zeilen mit den `INSERT INTO...`-Anweisungen, die wir in Listing 3.2 als Ausschnitt der Datei *Kurse.txt* gesehen haben. Sie sehen jetzt das unmittelbare Ergebnis dieser SQL-Anweisungen in der Tabelle `tbKurs`.

Abbildung 3.26
Inhalt der Tabelle tbKurs in mysql.exe




```
+-------+-------------------+----------+--------------+-----------------+------------+------------+
| KID   | Kurskennung       | KTHID    | KursdauerStunden | Kursbeginn | Kursende   |
| Zertifikat | Gebuehr | Ustpflichtig | DID  |
+-------+-------------------+----------+--------------+-----------------+------------+------------+
| CE17  | Celle17-Word      |        4 |              40 | 2007-04-23 | 2007-04-27 |
| eigen      |     280 | J            |  833 |
| CE20  | Celle20-Word      |        4 |              40 | 2007-07-09 | 2007-07-13 |
| eigen      |     280 | J            |  812 |
| CE23  | Celle23-Access    |        5 |              36 | 2007-08-06 | 2007-08-10 |
| eigen      |     350 | N            |  812 |
| CE24  | Celle24-Access    |        7 |              40 | 2007-08-13 | 2007-08-17 |
| eigen      |     400 | N            |  812 |
| H89   | Hannover89-Access |        5 |              32 | 2007-07-09 | 2007-07-13 |
| eigen      |     350 | N            |  812 |
| H90   | Hannover90-Windows|        1 |              18 | 2007-07-16 | 2007-07-18 |
| kein       |     120 | J            |  834 |
| H91   | Hannover91-Windows|        2 |              18 | 2007-07-23 | 2007-07-25 |
| kein       |     120 | J            |  834 |
+-------+-------------------+----------+--------------+-----------------+------------+------------+
7 rows in set (0.36 sec)

mysql>
```

Abbildung 3.27
Inhalt der Tabelle tbKurs in MySQL Query Browser

K...	Kurskennung	KTHID	KursdauerStunden	Kursbeginn	Kursende	Zertifikat	Gebuehr	Ustpflichtig	DID
CE17	Celle17-Word	4	40	2007-04-23	2007-04-27	eigen	280	J	833
CE20	Celle20-Word	4	40	2007-07-09	2007-07-13	eigen	280	J	812
CE23	Celle23-Access	5	36	2007-08-06	2007-08-10	eigen	350	N	812
CE24	Celle24-Access	7	40	2007-08-13	2007-08-17	eigen	400	N	812
H89	Hannover89-Access	5	32	2007-07-09	2007-07-13	eigen	350	N	812
H90	Hannover90-Windows	1	18	2007-07-16	2007-07-18	kein	120	J	834
H91	Hannover91-Windows	2	18	2007-07-23	2007-07-25	kein	120	J	834

Sie können die Eingabe einer SQL-Anweisung jetzt noch mit der ebenfalls bereits aus der zeilenorientierten Oberfläche bekannten Anweisung

```
SELECT * FROM tbKurs;
```

üben, was hoffentlich in MySQL Query Browser im Ergebnis zur Anzeige aller Datensätze der Tabelle `tbKurs` führt (siehe Abbildung 3.27). Denken Sie an das Ausführen der Anweisung und daran, dass die auszuführende Anweisung markiert sein muss, sonst erhalten Sie die Fehlermeldung:

```
Sie haben versucht, einen leeren String auszuführen. Bitte geben Sie
eine SQL-Anweisung in das Bearbeitungsfeld ein und führen Sie sie dann
aus.
```

Damit wird die Beschreibung der grundsätzlichen Logik von MySQL hier zunächst abgeschlossen. Sie sollten jetzt den grundsätzlichen Umgang beherrschen und mit Details wollen wir uns im Zusammenhang mit den SQL-Anweisungen beschäftigen. Wir wollen jetzt die restlichen Übungstabellen aufbauen, soweit nicht schon geschehen.

Datenimport

Nachdem Sie die Daten für die erste Tabelle erfolgreich importiert haben, können Sie mit allen weiteren Tabellen genauso verfahren:

```
mysql.exe -uroot -pmasterkey kurse < person.txt
mysql.exe -uroot -pmasterkey kurse < kursbesuche.txt
mysql.exe -uroot -pmasterkey kurse < dozent.txt
mysql.exe -uroot -pmasterkey kurse < kursthema.txt
```

Damit haben Sie die komplette Übungsdatenbank `kurse` in Ihrem MySQL angelegt. Die Funktionsfähigkeit prüfen Sie am besten mithilfe der grafischen Benutzeroberfläche. Geben Sie beispielhaft die SQL-Anweisung

```
SELECT * FROM tabellenname;
```

ein und führen Sie sie aus.

Tipp

Wenn Sie im Query Browser rechts doppelt auf die Tabelle klicken, wird automatisch die passende SQL-Anweisung erzeugt. Sie brauchen sie nur noch auszuführen. Nach der Ausführung können Sie ganz unten die Anzahl ermittelter Datensätze ablesen, beispielsweise „10 Zeilen in 0,0178 s (0,0004 s) geholt".

Sie sollten in `tbDozent` 5 Datensätze, in `tbKurs` 7 Datensätze, in `tbKursbesuche` 18 Datensätze, in `tbKursthema` 11 Datensätze und in `tbPerson` 20 Datensätze finden. Wenn Sie den Inhalt genau überprüfen wollen, sollten Sie Ihre Datenbankinhalte mit den im Anhang abgedruckten Tabellen vergleichen.

Import

Für größere Importe von Daten bietet MySQL noch das zusätzliche Utility *mysqlimport.exe*. Damit können Sie größere Datenmengen komfortabel in die Datenbank einfügen. Insbesondere können tabellarisch aufbereitete Daten sehr schnell eingelesen werden. Diese Daten müssen auch nicht zwingend auf dem Client liegen, sondern können auch direkt auf dem Server liegen. Im Prinzip führt *mysqlimport.exe* die SQL-Anweisung `LOAD DATA INFILE` aus. Wir wollen darauf im Rahmen der Administration und der Utilitys zurückkommen. Für unsere kleinen Datenmengen der Übungsdatenbanken und zum Verständnis der SQL-Logik reicht das `INSERT INTO` aus.

3.4 MS Access

3.4.1 Gründe für die Nutzung

MS Access ist eines der Datenbanksysteme, die in diesem Buch schwerpunktmäßig als Grundlage der Beispiele und Übungen verwendet werden. Im Fall von MS Access sind die Beispiele für MS Access 2003 erstellt – die zurzeit meistverbreitete Version.

Ältere MS Access-Versionen

Für die Nutzer älterer Versionen von MS Access (97, 2000) ist hier keine komplette Datenbank beigefügt. Sie können aber, wie die Nutzer aller Versionen von MS Access, eine eigene Datenbank anlegen, indem Sie die beigefügten Excel-Dateien importieren. Das Verfahren ist in Anhang A unter MS Access ausführlich beschrieben.

Access 2007

Für die Nutzer von MS Access 2007 ist die beigefügte Datenbank nutzbar. Sie können sie bei Bedarf auch konvertieren. Bei der Oberfläche gibt es natürlich ein paar Unterschiede, die Sie aber beim Umgang mit SQL nicht besonders behindern sollten.

Gründe für MS Access

MS Access wird hier als eines der Datenbankmanagementsysteme für die Übungen verwendet, obwohl es in vielen Bereichen etwas untypisch ist und von erfahrenen Datenbankanwendern als „kleine Münze“ eingestuft wird. Man sieht da oft so ein virtuelles Naserümpfen. Gleichwohl ist es die Datenbank, mit der viele Anwender als Erstes in Berührung kommen, einfach weil sie als Teil der MS Office-Umgebung von Microsoft den Weg zu vielen Anwendern findet. Es darf auch nicht übersehen werden, dass viele Anwender in Fachabteilungen und auch private Nutzer zunächst MS Access zur Verfügung haben. Die grafische Windows-Oberfläche erleichtert außerdem den Zugang zur Datenbankwelt. Drei wesentliche Gründe haben also dafür gesprochen, trotz aller Probleme und Besonderheiten gerade auch MS Access als eines der Datenbanksysteme zu verwenden, die die Grundlage für die Beispiele in diesem Buch bilden:

- Viele Leser haben MS Access als Übungsdatenbanksystem zur Verfügung. Es ist eines der gängigsten Datenbanksysteme.
- MS Access bietet durch seine Windows-Oberfläche eine vertraute Umgebung, in der viele Arbeiten in gewohnter Umgebung ablaufen können.
- Viele Anwendungen in den Unternehmen entstehen in Fachabteilungen und hier spielt MS Access wegen des breiten Office-Einsatzes eine wichtige Rolle.

Daher werden wir an vielen Stellen gezielt auf die Besonderheiten der grafischen Oberfläche von MS Access eingehen, um den interessierten Anwendern auch deren Nutzung nahezubringen und insbesondere den Zusammenhang zwischen der grafischen Oberfläche und dem darunterliegenden SQL zu erläutern. Dieser Zusammenhang kann auch für diejenigen interessant sein, die nicht MS Access nutzen, da die Struktur der Anweisungen manchmal auch in ihrer Umsetzung in der grafischen Windows-Oberfläche transparenter wird.

3.4.2 Eine Beispieldatenbank aufbauen

MDB-Datei

Auf die Installation soll hier nicht eingegangen werden. Sie erfolgt im Rahmen der Installation des Office-Paketes. Eine Datenbank für das Datenbankmanagementsystem MS Access besteht aus einer einfachen Datei im Windows-Dateisystem. Ähnlich einer Word- oder Excel-Datei wird für die komplette Datenbank eine Datei angelegt, die die Endung *.mdb* (Microsoft Data Base) erhält.

Tipp

Sie können MS Access-Datenbanken als Ganzes kopieren, verschieben oder löschen, indem Sie einfach die MDB-Datei kopieren, verschieben oder löschen. Achten Sie aber vor einer solchen Aktion stets darauf, dass die Datenbank nicht geöffnet ist, also kein Benutzer damit arbeitet. Dies können Sie leicht daran erkennen, dass keine Datei mit Sperren (Locks) existiert. Diese Datei hat denselben Namen wie die MDB-Datei, besitzt aber die Namenserweiterung *.lck*.

Nach dem Start von MS Access können Sie zunächst eine bestehende Datenbank öffnen oder eine neue Datenbank anlegen, wenn Sie DATEN/ÖFFNEN wählen.

Wenn Sie von der mitgelieferten CD aus dem Verzeichnis *datenbanken\MSAccess\kurse* die Datenbankdatei *Kurse2003.mdb* auf Ihre Festplatte kopieren, können Sie schnell starten. Wenn Sie die Datei direkt mit dem Windows-Explorer kopieren, müssen Sie darauf achten, den Schreibschutz zu entfernen, bevor Sie auf die Kopie zugreifen. Dies geschieht im Windows-Explorer durch Anklicken mit der rechten Maustaste, Auswahl der EIGENSCHAFTEN und Entfernen des Hakens für den Schreibschutz.

Öffnen Sie jetzt die Datenbank, indem Sie in MS Access entweder beim Start eine bestehende Datenbank auswählen oder nach dem Start mit DATEI/ÖFFNEN eine Datenbank auswählen. In beiden Fällen müssen Sie zu der Stelle Ihrer kopierten Datei *Kurse2003.mdb* navigieren und diese dann auswählen.

Alternativ können Sie natürlich auch im Windows-Explorer auf die Datei navigieren und sie mit einem Doppelklick starten. Bei der Installation von MS Access wurde die Erweiterung *.mdb* mit MS Access verbunden und führt zum Start des Datenbanksystems.

Sollten Sie beim Start eine Sicherheitswarnung erhalten, können Sie diese ignorieren und die Datenbank öffnen.

Sie können dann links TABELLEN wählen und sollten die fünf Tabellen der Kursdatenbank sehen. Mit einem Doppelklick können Sie deren Inhalt jeweils direkt betrachten.

In der Datenbank stehen die Tabellen nicht nur für sich allein, sondern sind durch Beziehungen miteinander verknüpft. Das ist nicht zwingend erforderlich, erleichtert aber oft die spätere Arbeit. Wählen Sie zur Ansicht der Beziehungen entweder im Hauptmenü EXTRAS/BEZIEHUNGEN oder das entsprechende Symbol in der Titelleiste.

Beziehungen

Sie bekommen alle Tabellen angezeigt. Verschieben Sie sie und vergrößern Sie sie bei Bedarf so, dass Sie beispielsweise das Ergebnis aus Abbildung 3.1 erreichen. Sie sehen in der Abbildung, dass jeweils zwei Tabellen durch eine Beziehung verbunden sind. Diese Beziehungen enden jeweils auf Höhe eines bestimmten Datenfeldes, beispielsweise ist das Datenfeld `PID` in der Tabelle `tbPerson` mit dem Datenfeld `KTID` in der Tabelle `tbKursbesuche` verbunden. Derartige Felder bilden den Primärschlüssel der einen Tabelle und den Fremdschlüssel der anderen Tabelle.

Sie können eine solche Beziehung im Prinzip auf zwei Arten anlegen. Entweder Sie „greifen“ mit der linken Maustaste eines der beiden Felder und ziehen es mit gedrückter Maustaste auf das andere Feld. Sie sehen an der Veränderung des Cursors, wann eine Verbindung zustande kommen kann. Hier lassen Sie das Feld einfach los. Haben Sie sich vertan, markieren Sie die Verbindungslinie und löschen die Verbindung mit der Taste Entf.

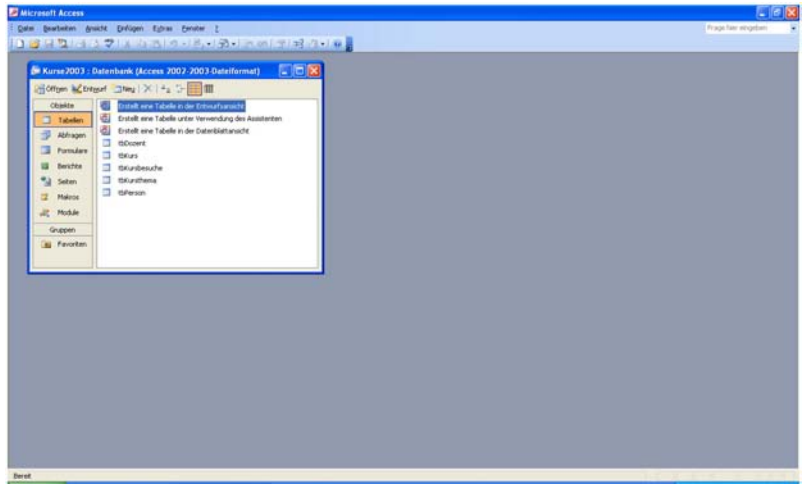
Die zweite Möglichkeit ist die Funktion BEZIEHUNGEN/BEZIEHUNGEN BEARBEITEN. Hier erhalten Sie ein Fenster wie in Abbildung 3.28 dargestellt. Sie müssen die an der Beziehung beteiligten Tabellen sowie das Primärschlüssel- und das Fremdschlüsselfeld auswählen. Beachten Sie, dass die Auswahl nicht symmetrisch ist. Sie müssen links als Tabelle die Tabelle mit dem Primärschlüsselfeld angeben. Rechts wird als Detailtabelle die Tabelle mit dem Fremdschlüsselfeld ausgewählt.

Abbildung 3.28
Aufbau einer Beziehung zwischen tbPerson und tbKursbesuche in MS Access

In der Datenbank sind die Beziehungen bereits eingetragen. Sie sollten Sie jetzt nicht inhaltlich verändern. Schließen Sie das Beziehungsfenster wieder und speichern Ihre eventuellen grafischen Veränderungen dabei ab.

Damit haben Sie die Kursdatenbank als Musterdatenbank `kurse` vorerst komplett angelegt.

Abbildung 3.29
Datenbankfenster nach dem Import



3.5 Oracle

3.5.1 Gründe für die Nutzung

Oracle ist eines der „großen“ Datenbankmanagementsysteme und neben DB2 das Datenbanksystem, das gerade in Großunternehmen oft das Rückgrat der IT bildet. Oracle ist inzwischen in verschiedenen Versionen in jeder Größenordnung der IT bis hinunter zur Einzelplatzversion für Windows verfügbar und hier auch für nicht kommerzielle Zwecke testweise kostenlos nutzbar. Daher ist es auch hier interessant, es zu nutzen. Oracle war immer wieder Vorreiter bei der Umsetzung komplexer SQL-Erweiterungen, was einerseits für umfangreiche Anwendungen sehr interessant ist, andererseits natürlich umgekehrt aber auch mit dem Nachteil verbunden ist, dass vieles was in Oracle „geht“ in anderen Systemen nicht möglich ist. Wer Oracle bereits kennt oder entscheidet, dass ihn gerade wegen seiner Bedeutung für größere Unternehmen Oracle besonders interessiert, kann die Beispiele auch mit dem beiliegenden *Oracle Express* bearbeiten.

3.5.2 Oracle installieren

Zunächst soll hier kurz die Installation der Oracle 10g Express Edition erläutert werden, für die Sie die Hinweise zum Download unter *www.serval.de/SQL* finden. Starten Sie die Datei auf Ihrem PC. Damit startet der INSTALLATIONS-ASSISTENT (siehe Abbildung 3.30).

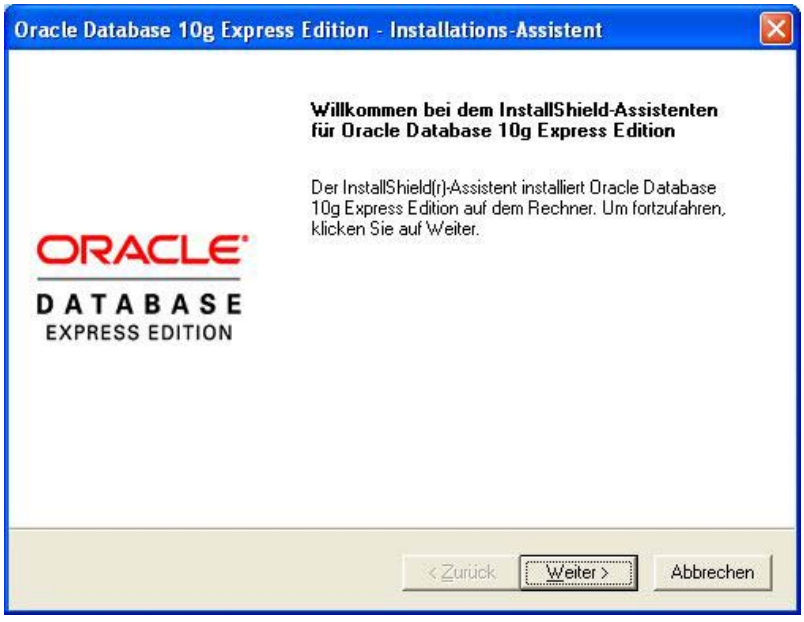
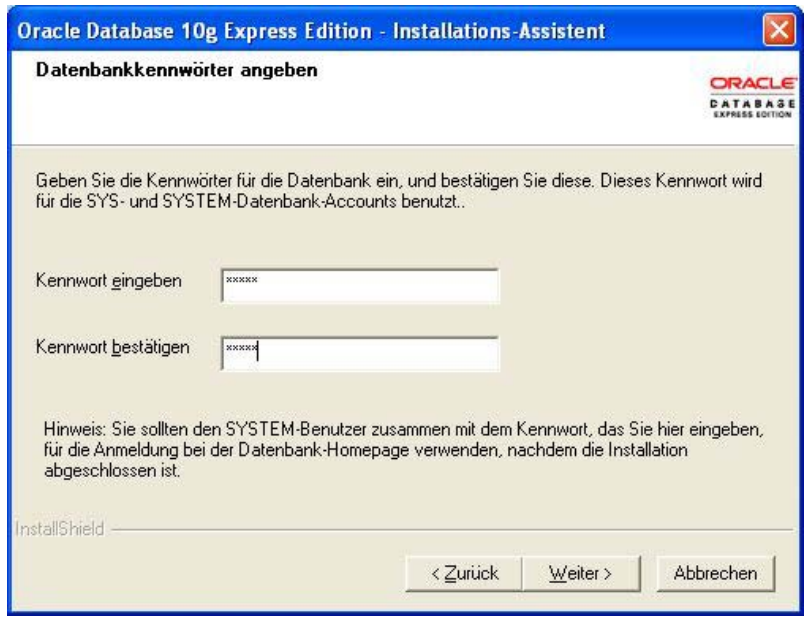

Oracle Database 10g Express Edition - Installations-Assistent
Willkommen bei dem InstallShield-Assistenten für Oracle Database 10g Express Edition
Der InstallShield(r)-Assistent installiert Oracle Database 10g Express Edition auf dem Rechner. Um fortzufahren, klicken Sie auf Weiter.
ORACLE
DATABASE
EXPRESS EDITION
< Zurück
Weiter >
Abbrechen

Abbildung 3.30
Startbildschirm des Oracle-Installations-Assistenten

SYS und SYSTEM

Nach den gängigen Fragen zur Lizenzierung und zum Zielverzeichnis für die Installation fragt Sie Oracle nach einem Passwort. Dieses Passwort wird gleich für zwei Superuser verwendet, die Oracle *SYS* und *SYSTEM* nennt. Die genauen Unterschiede sind hier nicht relevant, im Wesentlichen wird mit SYSTEM gearbeitet, während SYS ein „Backup" für den Administrator darstellt. Verwenden Sie als Passwort am besten wieder *masterkey*, wie in diesem Buch immer empfohlen und worauf auch die Skripte abgestimmt sind.

Abbildung 3.31
Eingabe des Passwortes für die Superuser


Oracle Database 10g Express Edition - Installations-Assistent
Datenbankkennwörter angeben
ORACLE
DATABASE
EXPRESS EDITION
Geben Sie die Kennwörter für die Datenbank ein, und bestätigen Sie diese. Dieses Kennwort wird für die SYS- und SYSTEM-Datenbank-Accounts benutzt..
Kennwort eingeben
*****
Kennwort bestätigen
*****
Hinweis: Sie sollten den SYSTEM-Benutzer zusammen mit dem Kennwort, das Sie hier eingeben, für die Anmeldung bei der Datenbank-Homepage verwenden, nachdem die Installation abgeschlossen ist.
InstallShield
< Zurück
Weiter >
Abbrechen

Danach läuft die Installation des Oracle-Servers als Dienst ab. Sie bekommen die drei wesentlichen Ports angezeigt, die Oracle verwendet. Diese führen zumeist nicht zu größeren Problemen mit der Firewall, wenn doch, finden Sie im Rahmen der Beschreibung der MySQL-Installation eine Beschreibung der Vorgehensweise zur Freigabe einzelner Ports in der Firewall.

Nach der erfolgten Installation des Servers, die etwas Geduld Ihrerseits erfordert, aber keine weiteren Besonderheiten oder wichtigen Auswahlmöglichkeiten aufweist, können Sie über die normale Windows-Start-Schaltfläche eine Reihe von Funktionen aufrufen (siehe Abbildung 3.32).

Sie sehen, dass Sie den Oracle-Server hier starten und stoppen können, was den Vorteil hat, dass alle Dienste koordiniert werden. Für den Zugriff auf die Datenbank bietet Oracle die SQL-Befehlszeile und die Option GEHE ZU DATENBANK-HOMEPAGE als grafische Oberfläche an. Auf die Erläuterung der Befehlszeile soll hier verzichtet werden, da dies den Rahmen sprengen würde.

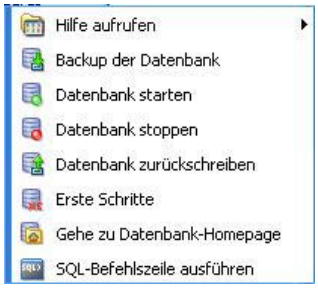


Abbildung 3.32
Oracle-Werkzeuggruppe

Wenn Sie zu der angebotenen Datenbank-Homepage gehen, öffnet sich Ihr Standard-Browser. Die Oberfläche besteht also aus einer Browser-Anwendung. Sie können sich jetzt mit dem Benutzer *SYSTEM* und dem von Ihnen während der Installation angegebenen Passwort anmelden. Danach kommen Sie auf eine Übersichtsseite wie sie die Abbildung 3.33 darstellt.

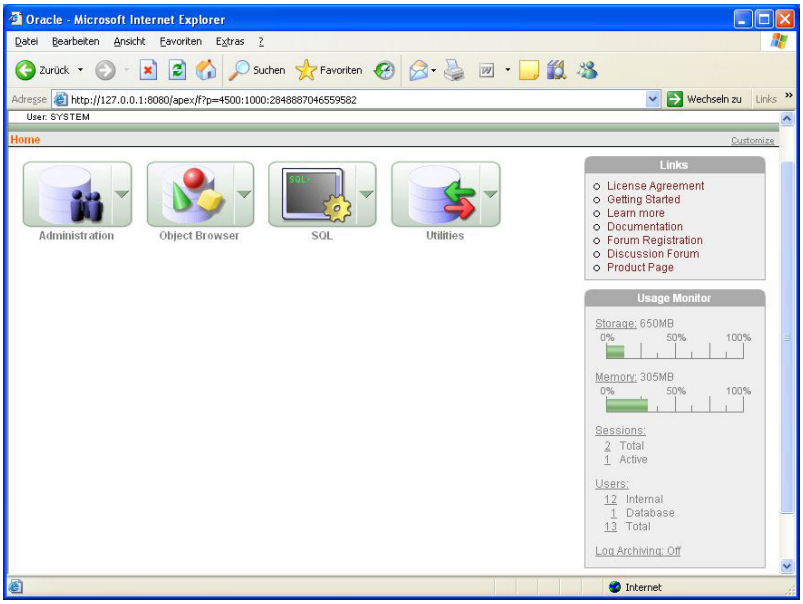


Abbildung 3.33
Übersichtsfenster der Oracle-Umgebung auf Browser-Basis

Der Bereich ADMINISTRATION dient der kompletten Speicher- und Benutzerverwaltung. Im OBJECT BROWSER können Sie sich die Datenbankstruktur und die Inhalte überblickartig anzeigen lassen, während der Bereich SQL der Eingabe von SQL-Anweisungen dient und eine zentrale Rolle spielen wird. Die weiteren UTILITIES sollen uns hier nicht interessieren. Die einzelnen Bereiche sollen beschrieben werden, wenn wir sie nutzen, was mit der Installation der Beispieldatenbank *kurse* beginnt. Zum Thema „Oracle“ finden Sie im Übrigen in Anhang A eine kurze Beschreibung der Funktionsweise.

3.5.3 Die Testdatenbanken importieren

Oracle bietet eine Fülle von Möglichkeiten, um Daten in die Datenbank zu laden. Flexibilität, Performance, Zeichensatzprobleme und Ähnliches bieten hier ein breites Spektrum für Vergleiche. In unserem Rahmen soll nur der einfache Import einer vergleichsweise kleinen Datenmenge im Mittelpunkt stehen, wozu einfache Skripte genutzt werden, die mit dem mitgelieferten *SQLPlus* importiert werden.

Im ersten Schritt soll das Schema für die Kurs-Beispieldatenbank erzeugt werden. Dazu ist es ausreichend, wenn Sie aus dem CD-Verzeichnis *\Datenbanken\Oracle\kurse* das Skript *OraKurse.bat* durch einen Doppelklick starten. Es wird das ORACLE-Werkzeug SQLPlus gestartet. Haben Sie nicht *masterkey* als Passwort verwendet, müssen Sie das Skript kopieren und entsprechend anpassen. Danach läuft das Skript automatisch ab und schließt das Fenster wieder.

Testen Sie den erfolgreichen Ablauf, indem Sie sich über die Browser-Oberfläche anmelden. Verwenden Sie jetzt als Username *kurse* mit dem Passwort *pwkurse*. Hat der Import funktioniert, können Sie über den Objekt-Browser das Ergebnis kontrollieren.

Durch Ausführung des Skripts haben Sie ein Datenbankschema erzeugt, das wie unser Benutzer `kurse` heißt. Sie können dieses Schema jetzt inhaltlich kontrollieren, indem Sie das Auswahlmenü des OBJECT BROWSER öffnen, auf BROWSE und dann auf TABLES verzweigen (siehe Abbildung 3.34).

Abbildung 3.34
Der Object Browser erlaubt die Kontrolle des Inhaltes.

Sie kommen in das Übersichtsfenster des Browsers. Hier sind zunächst die fünf Tabellen des Schemas für die Kursdatenbank vorhanden (siehe Abbildung 3.35). Sie können jeweils auf die Tabellennamen klicken, um sich die Struktur der einzelnen Tabellen anzusehen.

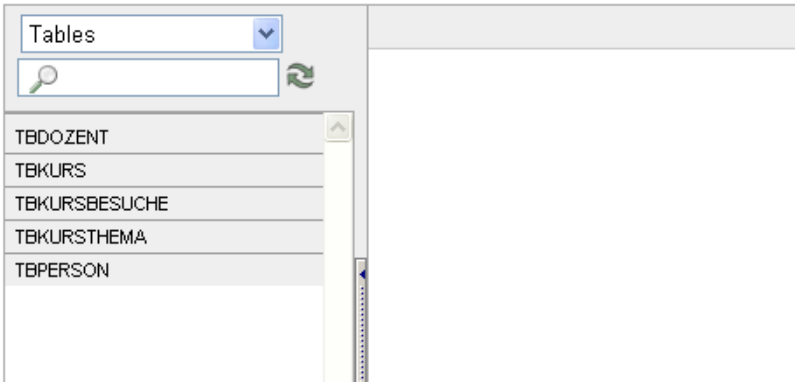


Abbildung 3.35
Tabellen des Schemas kurse

Fehlerfall

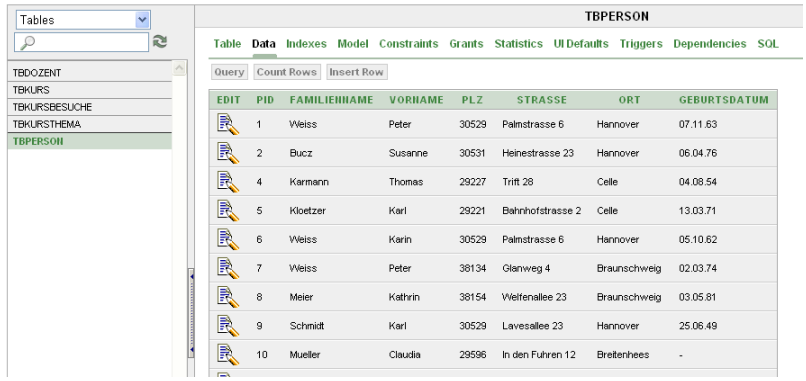
Sollte die Anmeldung nicht funktionieren oder sollten keine Tabellen vorhanden sein, öffnen Sie die Datei *CreateKurse.sql* im *Oracle*-Ordner unter *Datenbanken* auf Ihrer CD mit einem normalen Editor wie dem *Notepad*. Entfernen Sie die letzte Anweisung `exit` aus der Datei und speichern Sie sie neu auf Ihrer Festplatte ab. Versuchen Sie danach erneut das Skript *OraKurse.bat* zu starten. Sie sollten jetzt nach Eingabe des Passwortes die Fehlermeldungen sehen, die das Problem verursacht haben. Analysieren Sie sie insbesondere im Hinblick auf Benutzer, Passwort und eventuell Verzeichnisse und versuchen Sie es erneut. Wahrscheinlich haben Sie ein falsches Verzeichnis oder ein falsches Passwort verwendet.

Datenimport

Neben der Erzeugung der Struktur sind durch den Aufruf des Skriptes auch die eigentlichen Daten importiert worden. Dazu dienen die weiteren Zeilen in *OraKurse.bat*, die einen Aufruf und eine Ausführung der SQL-Anweisungen in den Dateien *person.sql*, *kursthema.sql*, *dozent.sql*, *kurse.sql* und *kursbesuche.sql* bewirken.

Den Erfolg können Sie wieder im Objekt-Browser überprüfen. Nach Auswahl einer Tabelle im linken Teil des Fensters (siehe Abbildung 3.36) können Sie rechts zwischen verschiedenen Darstellungen wählen. Die Basisdarstellung TABLE, die die Struktur der Tabelle zeigt, kennen Sie bereits. Die zweite Darstellung, DATA, unmittelbar daneben gibt die in die Tabelle geladenen Daten wieder. Die weiteren Darstellungen geben Strukturen an, die erst später besprochen werden. Interessant ist noch die letzte Darstellung, SQL, die die SQL-Anweisung wiedergibt, mit der die Tabelle erzeugt wurde. Sie können dies jeweils mit der Angabe in der Datei *CreateKurse.sql* vergleichen, die wir zum Aufbau der Struktur verwendet haben.

Abbildung 3.36
Tabelle Person


Tables
TBDOZENT
TBKURS
TBKURSBESUCHE
TBKURSTHEMA
TBPERSON
TBPERSON
Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL
Query Count Rows Insert Row
EDIT PID FAMILIENNAME VORNAME PLZ STRASSE ORT GEBURTSDATUM
1 Weiss Peter 30529 Palmstrasse 6 Hannover 07.11.63
2 Bucz Susanne 30531 Heinestrasse 23 Hannover 06.04.76
4 Karmann Thomas 29227 Trift 28 Celle 04.08.54
5 Kloetzer Karl 29221 Bahnhofstrasse 2 Celle 13.03.71
6 Weiss Karin 30529 Palmstrasse 6 Hannover 05.10.62
7 Weiss Peter 38134 Glanweg 4 Braunschweig 02.03.74
8 Meier Kathrin 38154 Welfenallee 23 Braunschweig 03.05.81
9 Schmidt Karl 30529 Lavesallee 23 Hannover 25.06.49
10 Mueller Claudia 29596 In den Fuhren 12 Breitenhees -

Die komplette Struktur der Tabellen mit ihren Beziehungen ist der Abbildung 3.1 zu entnehmen.

Import-Utility

Für den Import umfangreicher Daten kann in Oracle das Original-Import-Utility *imp.exe* verwendet werden. Oracle bietet seit der Version 10 auch die Datenpumpe *impdp.exe* an. Da aber eventuell ältere Oracle-Versionen zum Einsatz kommen, wird diese hier noch nicht verwendet. Für unsere Zwecke reicht aber der Weg über die konventionellen SQL-Anweisungen aus, wie er hier beschrieben wurde.

Weitere Hinweise zur Benutzung der Oracle-Oberfläche finden Sie in Anhang A.

Abfragen können Sie in Oracle ausführen, indem Sie auf der HOME-Seite das Auswahlmenü zur SQL-Schaltfläche aufklicken und den Befehl SQL COMMANDS/ENTER COMMAND auswählen. Danach können Sie im oberen Fenster beispielsweise

```
SELECT * FROM tbPerson;
```

eingeben und anschließend die Schaltfläche RUN verwenden. Im unteren Bereich erscheint dann das Ergebnis.

3.6 Firebird

Firebird ist der Nachfolger und die Weiterentwicklung des Borland-Datenbankmanagementsystems *Interbase*. Es handelt sich wiederum um eine für Testzwecke frei nutzbare Datenbank, die dank der Vorarbeiten in Interbase eine erstaunliche Stärke und Robustheit für den produktiven Einsatz aufweist. Die geringe Verbreitung ist sicher in erster Linie auf ihre Herkunft aus dem Haus Borland, jetzt CodeGear (Embarcadero), zurückzuführen. Der Ruf von Borland ist der eines hervorragenden Lieferanten von Softwareentwicklungsumgebungen wie Delphi, C++-Builder und inzwischen einer Reihe von

Umgebungen für die Internetprogrammierung, aber eben nicht der eines Datenbankherstellers. Das Schattendasein des Systems ist daher eher auf die fehlende Positionierung im Markt als auf technische Schwächen zurückzuführen.

Das System wurde hier trotz der vergleichsweise geringen Verbreitung auch aufgenommen, da es gerade für Programmierer von Anwendungen mit einer embedded Database eine interessante Alternative bietet. Firebird bietet einen vergleichsweise geringen Funktionsumfang, der sich aber mithilfe sogenannter User Defined Functions (UDF) schnell erweitern lässt. Das System ist sehr schlank und stabil. Es ist aber als eigenständige Datenbank mit grafischer Oberfläche eher untypisch.

Damit Sie dieses System ebenfalls für die Beispiele des Buches nutzen können, wird kurz die Installation und die Bereitstellung der Beispieldatenbanken beschrieben. Die Beispiele sind aber – ähnlich wie bei Oracle – nicht schwerpunktmäßig auf Firebird abgestimmt, sodass einige Beispiele wegen der etwas unterschiedlichen Philosophie von Firebird mangels vorinstallierter Funktionen nicht genutzt werden können.

Sie finden alle Hinweise zum Download unter *www.serval.de/SQL*. Starten Sie die Installation der Firebird-Version 2.0.3 mit der entsprechenden Datei *Firebird-2.0.3.12981-1-Win32.exe*. Der Installationsassistent meldet sich mit dem Logo in der Abbildung 3.37.

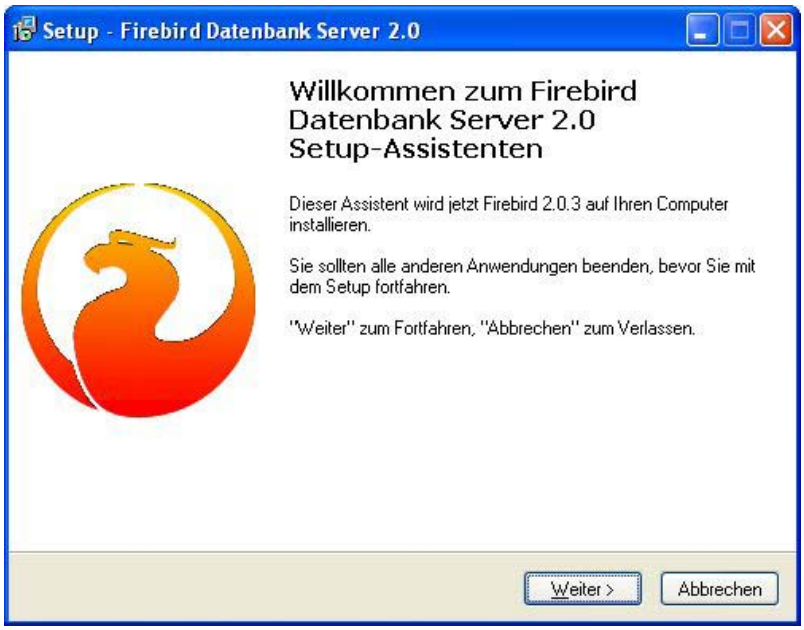
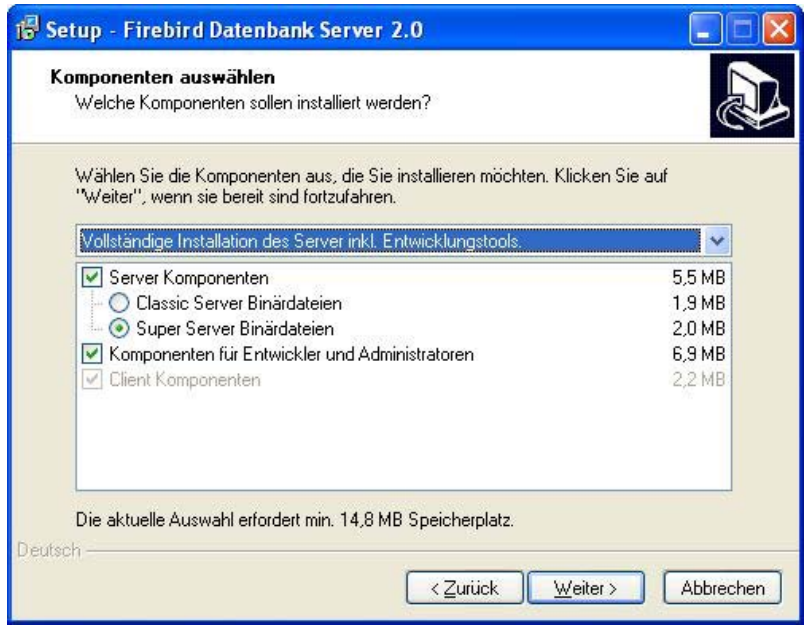

Setup - Firebird Datenbank Server 2.0
Willkommen zum Firebird Datenbank Server 2.0 Setup-Assistenten
Dieser Assistent wird jetzt Firebird 2.0.3 auf Ihren Computer installieren.
Sie sollten alle anderen Anwendungen beenden, bevor Sie mit dem Setup fortfahren.
"Weiter" zum Fortfahren, "Abbrechen" zum Verlassen.
Weiter >
Abbrechen

Abbildung 3.37
Start des Firebird-Installationsassistenten

Sie können jetzt wie üblich durch die Installation gehen, indem Sie zunächst die Lizenzvereinbarung lesen. Der folgende Hinweis auf die geänderte Struktur ist nur für die bisherigen Nutzer des Systems relevant. Installieren Sie das System das erste Mal, können Sie den Hinweis ignorieren, alle Dateien sind auf die Version 2 abgestimmt.

Abbildung 3.38
Wahl des Super Servers


Setup - Firebird Datenbank Server 2.0
Komponenten auswählen
Welche Komponenten sollen installiert werden?
Wählen Sie die Komponenten aus, die Sie installieren möchten. Klicken Sie auf "Weiter", wenn sie bereit sind fortzufahren.
Vollständige Installation des Server inkl. Entwicklungstools.
Server Komponenten 5,5 MB
Classic Server Binärdateien 1,9 MB
Super Server Binärdateien 2,0 MB
Komponenten für Entwickler und Administratoren 6,9 MB
Client Komponenten 2,2 MB
Die aktuelle Auswahl erfordert min. 14,8 MB Speicherplatz.
Deutsch
< Zurück
Weiter >
Abbrechen

Wenn Sie weiter dem Installationsassistenten folgen, gelangen Sie schließlich zu der Abfrage in Abbildung 3.38. Wesentlich ist die Wahl der CLASSIC SERVER- oder der SUPER SERVER-Komponenten. Wegen der besseren Unterstützung ist unter Windows in jedem Fall der SUPER SERVER vorzuziehen.

Die weitere Auswahl ist in Abbildung 3.39 angegeben. Der Guardian ist ein nützliches Hilfsmittel zur Kontrolle des Servers. Die Installation als Server folgt der Logik der anderen Datenbanken. Der automatische Start des Servers bei Systemstart sollte wiederum je nach Häufigkeit der Nutzung entschieden werden.

Nach einigen weiteren Standardeinstellungen ist die Installation abgeschlossen und Firebird bestätigt dies, wie in Abbildung 3.40 zu sehen. Die Installation können Sie in der Windows-Systemsteuerung kontrollieren. Dort müsste bei Ihnen als zusätzliches Symbol der „Guardian“ (Firebird Server Manager) auftauchen. Er dient der Kontrolle des Firebird-Servers.

Setup - Firebird Datenbank Server 2.0
Zusätzliche Aufgaben auswählen
Welche zusätzlichen Aufgaben sollen ausgeführt werden?
Wählen Sie die zusätzlichen Aufgaben aus, die das Setup während der Installation von Firebird Datenbank Server 2.0 ausführen soll, und klicken Sie danach auf "Weiter".
Den Guardian zum Kontrollieren des Servers verwenden?
Firebird Server starten als:
Server als Anwendung starten?
Server als Dienst starten?
Firebird nach jedem Rechnerneustart automatisch starten?
"Installieren eines Symbols im Infobereich der Taskleiste?"
Die Firebird Client-Bibliothek ins <system>-Verzeichnis kopieren?
Die Client-Bibliothek als GDS32.DLL ins <system>-Verzeichnis kopieren?
Deutsch
< Zurück
Weiter >
Abbrechen

Abbildung 3.39
Konfiguration der Firebird-Installation

Setup - Firebird Datenbank Server 2.0
Beenden des Firebird Datenbank Server 2.0 Setup-Assistenten
Setup hat die Installation von Firebird Datenbank Server 2.0 auf Ihren Computer abgeschlossen. Die Anwendung kann über die installierten Programm-Verknüpfungen gestartet werden.
Klicken Sie auf "Fertigstellen", um das Setup zu beenden.
Firebird jetzt als Dienst starten?
< Zurück
Fertigstellen

Abbildung 3.40
Abschluss der Installation

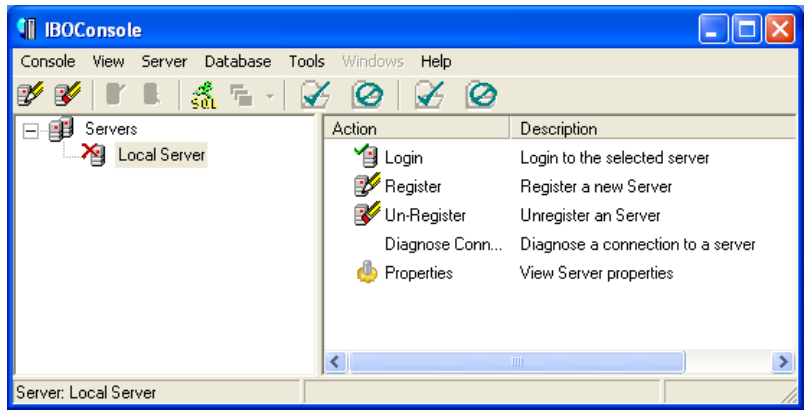
Firebird bietet ähnlich wie MySQL nicht direkt eine grafische Benutzeroberfläche. Allerdings gibt es wieder wie bei MySQL Werkzeuge anderer Anbieter. Eines dieser Werkzeuge der Firma IB, die *IBOConsole*, ist ebenfalls über

einen Verweis auf *www.serval.de/SQL* erreichbar und kann für Testzwecke genutzt werden. Die Installation besteht lediglich aus einem Entpacken des Werkzeugs in ein Verzeichnis Ihres Rechners.

Sie können danach die Datei *IBOConsole.exe* direkt starten. Auf Dauer ist es aber sinnvoller, einen Link auf dem Desktop oder im Programmsystem einzufügen, um den Aufruf zu vereinfachen.

Die Oberfläche der IBOConsole ist in Abbildung 3.41 erkennbar.

Abbildung 3.41
Anmeldung am Firebird-Server

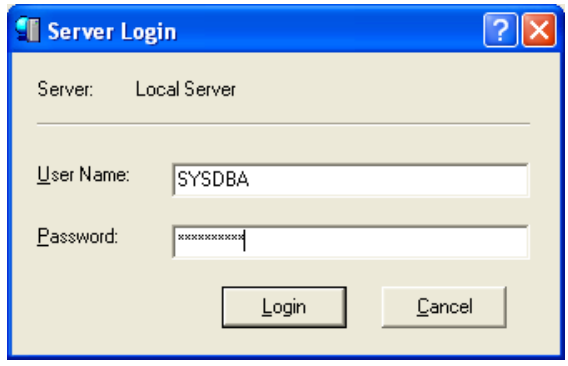


Melden Sie sich beim Server an, entweder über die Funktion im rechten Fensterteil (LOGIN) oder über die rechte Maustaste (jeweils nach Markierung des Servers).

SYSDBA

Nutzen Sie dazu den Superuser *SYSDBA* mit dem automatisch erzeugten Passwort *masterkey* (siehe Abbildung 3.42).

Abbildung 3.42
Anmeldung mit dem Superuser SYSDBA



Der Server öffnet sich und zeigt die enthaltenen Bestandteile an. Es sind jetzt, wie in Abbildung 3.43 zu sehen, noch keine Datenbanken vorhanden.

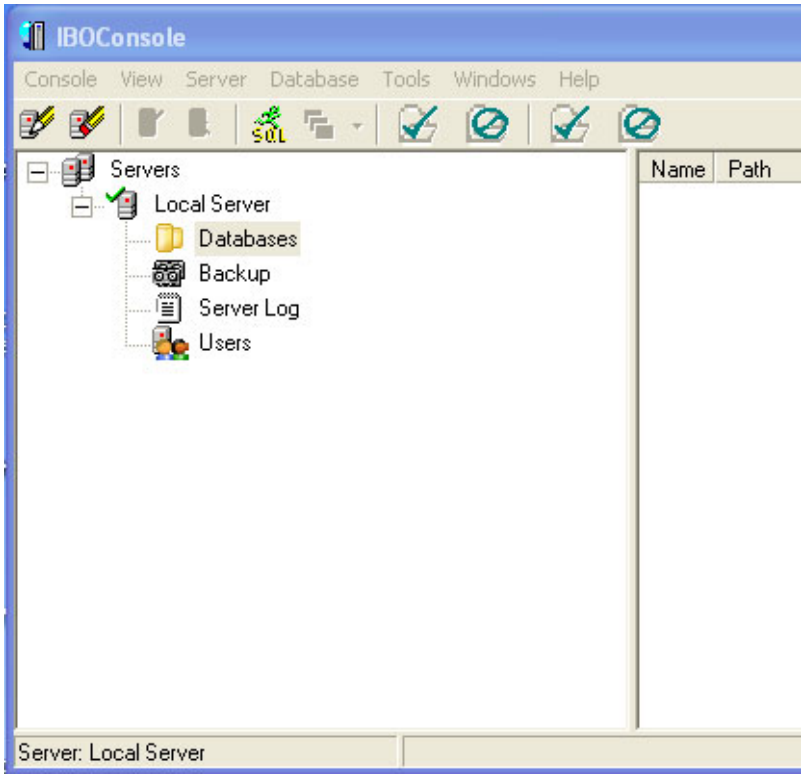


Abbildung 3.43
Nach der Anmeldung beim Server

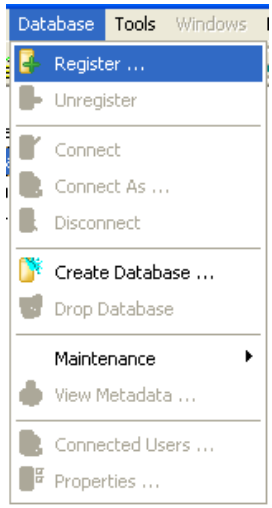
Jetzt können Sie eine Datenbank anlegen. Dazu finden Sie auf der CD unter *\Datenbanken\Firebird\kurse* die Datei *FireKurse.bat*. Diese Datei lädt die in *CreateKurse.sql* niedergeschriebenen SQL-Anweisungen in Firebird, um die Datenbank `kurse` zu erzeugen. Dabei wird eine Datei *Kurse.fdb* (Firebird Database) angelegt.

In der ersten Zeile finden Sie eine Pfadangabe für die Ablage der Datei. Diese entspricht der Standardinstallation. Prüfen Sie, ob der Pfad in der ersten Zeile für Ihr System gültig ist:

```
CREATE DATABASE 'C:\Programme\Firebird\Firebird_2_0\KURSE.fdb';
```

Diese Datei müssen Sie noch beim Server registrieren, damit er sie in seine Dateiverwaltung aufnimmt. Rufen Sie dazu den Menupunkt DATABASE/REGISTER in der Konsole auf.

Abbildung 3.44
Registrierung der Firebird-Datenbankdatei

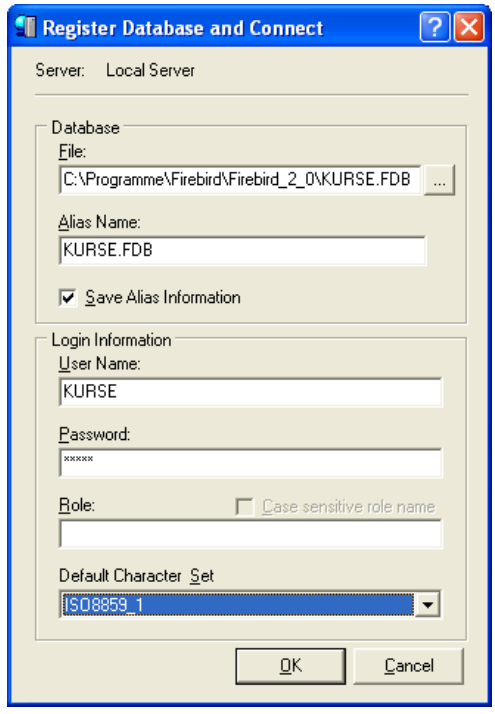


Klicken Sie auf die ...-Schaltfläche und folgen Sie im SELECT DATABASE-Dialog dem Pfad zu der oben angegebenen Datei, im Normalfall also

C:\Programme\Firebird\Firebird_2_0

und wählen Sie die Datei *kurse.fdb* aus (siehe Abbildung 3.45). Sie können noch eine eigene Anmeldung sowie einen Zeichensatz angeben.

Abbildung 3.45
Registrierung der Datenbank



Nach der Registrierung der Datenbank können Sie sich im Hauptfenster mit einem CONNECT anmelden und erhalten die Übersicht in Abbildung 3.46.

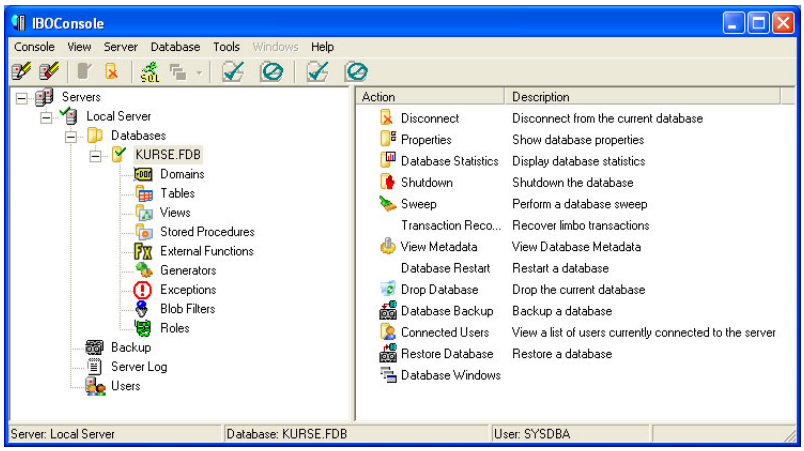


Abbildung 3.46
Übersicht über die Datenbank KURSE nach der Anmeldung

Wählen Sie beispielsweise TABLES, dann erhalten Sie die Übersicht in Abbildung 3.47.

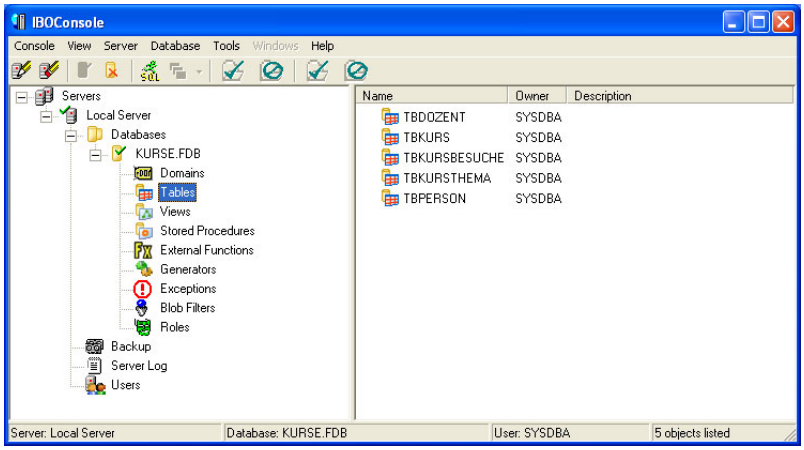


Abbildung 3.47
Tabellenübersicht

Hier können Sie sich jetzt über den aktuellen Stand informieren. Ein Hinweis soll an dieser Stelle noch erfolgen. Sollten Sie im Lauf der Arbeit wieder auf den aktuellen Stand kommen wollen, können Sie die Datenbank mit DROP DATABASE entfernen und dann wie beschrieben neu aufbauen. Das Kommando ist auch im Skript *LoescheFireKurse* enthalten. Führen Sie es bei geschlossener Datenbank aus und starten danach wieder *FireKurse*, so haben Sie den Ausgangszustand wiederhergestellt.

Abbildung 3.48
Entfernen der Datenbank

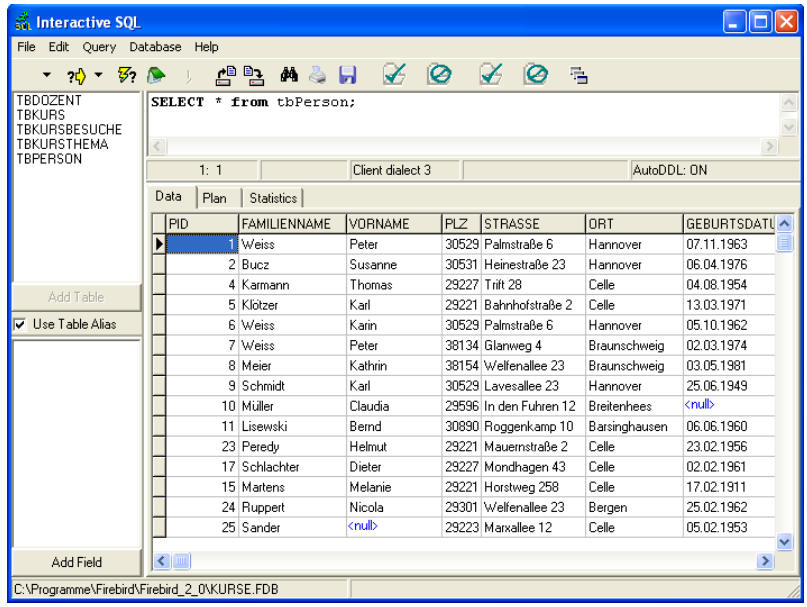
Wenn Sie eine SQL-Abfrage durchführen wollen, wählen Sie TOOLS/INTERACTIVE SQL aus.

Geben Sie im oberen Fenster einfach die SQL-Anweisung ein, beispielsweise

```
SELECT * from tbPerson;
```

und betätigen Sie die Schaltfläche mit dem gelben Blitz oder wählen Sie im Menu QUERY/EXECUTE. Sie sehen das Ergebnis der Abfrage im unteren Fenster. Es sollte etwa wie in Abbildung 3.49 aussehen.

Abbildung 3.49
Tabelle in Firebird


Interactive SQL
File Edit Query Database Help
TBDOZENT
TBKURS
TBKURSBESUCHE
TBKURSTHEMA
TBPERSON
SELECT * from tbPerson;
1: 1
Client dialect 3
AutoDDL: ON
Data Plan Statistics
PID FAMILIENNAME VORNAME PLZ STRASSE ORT GEBURTSDATU
1 Weiss Peter 30529 Palmstraße 6 Hannover 07.11.1963
2 Bucz Susanne 30531 Heinestraße 23 Hannover 06.04.1976
4 Karmann Thomas 29227 Trift 28 Celle 04.08.1954
5 Klötzer Karl 29221 Bahnhofstraße 2 Celle 13.03.1971
6 Weiss Karin 30529 Palmstraße 6 Hannover 05.10.1962
7 Weiss Peter 38134 Glanweg 4 Braunschweig 02.03.1974
8 Meier Kathrin 38154 Welfenallee 23 Braunschweig 03.05.1981
9 Schmidt Karl 30529 Lavesallee 23 Hannover 25.06.1949
10 Müller Claudia 29596 In den Fuhren 12 Breitenhees <null>
11 Lisewski Bernd 30890 Roggenkamp 10 Barsinghausen 06.06.1960
23 Peredy Helmut 29221 Mauernstraße 2 Celle 23.02.1956
17 Schlachter Dieter 29227 Mondhagen 43 Celle 02.02.1961
15 Martens Melanie 29221 Horstweg 258 Celle 17.02.1911
24 Ruppert Nicola 29301 Welfenallee 23 Bergen 25.02.1962
25 Sander <null> 29223 Marxallee 12 Celle 05.02.1953
Add Table
Use Table Alias
Add Field
C:\Programme\Firebird\Firebird_2_0\KURSE.FDB

Weitere Hinweise zum Umgang mit dem „Feuervogel“ finden Sie in Anhang A.

3.7 OpenOffice.orgBase

Abschließend sehen wir uns jetzt noch eine Alternative zum Einsatz von MS Access an, die auch viele Ähnlichkeiten mit der MS Access-Umgebung aufweist. So steht mit der Datenbankkomponente *orgBase* von OpenOffice 2.2 eine weitere Datenbank zur freien Nutzung für die hier benötigten Beispiele zur Verfügung. Die Einbindung in eine Office-Umgebung und die für viele Aufgaben unmittelbar nutzbare grafische Oberfläche sind MS Access nicht unähnlich. In der Art der Ausstattung und Nutzung, bei den Datentypen und auch beim SQL-Dialekt gibt es allerdings auch gravierende Unterschiede. So ähnelt MS Access oft mehr einer Entwicklungsumgebung für Datenbankanwendungen unter Windows und speziell unter MS Office, während openOffice.orgbase mehr Ähnlichkeiten mit den „großen Geschwistern“ MySQL oder Firebird aufweist. Auch wenn dies nicht ganz korrekt ist, wollen wir die Datenbank im Folgenden entsprechend der Oberfläche immer kurz als „openBase“ bezeichnen.

openBase wurde hier als „freie“ Alternative zu MS Access aufgenommen. Es ist neben MS Access hier die einzige Datenbank, die direkt mit einer grafischen Oberfläche ausgeliefert wird und somit unmittelbar kontrollierbar ist. Eine Datenbank in openBase ist analog der MS Access-Datei mit der Erweiterung *.mdb* eine einzelne Datei mit der Erweiterung *.odb*. Natürlich werden wir auch hier die direkte Eingabe der SQL-Anweisungen vorstellen, die Oberfläche bietet aber einen schnellen Überblick. Der Funktionsumfang ist sehr gut und stärker am Standard angelehnt als bei MS Access, dafür fehlt weitgehend die Funktionalität eines echten Multiuserbetriebes.

HSQL

Basis der Datenbank openBase ist die sogenannte HSQL-Engine. Daneben unterstützt openBase auch die Weiterleitung von Anweisungen an andere Datenbanksysteme. Hier wird aber die eigentliche und eigene HSQL-Engine für die Beispiele mit openBase verwendet. Unter dem entsprechenden Schlagwort finden Sie auch die meiste Hilfe im Internet.

Wenn Sie openBase als Beispielumgebung für die Übungen dieses Buches nutzen wollen, müssen Sie zunächst OpenOffice installieren. Die Version finden Sie auf der CD im Unterordner *OpenBase*.

Nach dem Start von *OOosetup.exe* meldet sich der Installationsassistent, der Ihnen ein Installationsverzeichnis vorschlägt. Sie können das komplette Paket oder nur openBase (*OpenOffice.org Base*) installieren. Für unsere Zwecke reicht die Base-Komponente aus. Spätestens, wenn Sie aber Daten importieren oder exportieren wollen, benötigen Sie zumindest noch die Calc-Komponente.

Die weiteren Fragen können Sie entsprechend Ihres Systems beantworten. Beachten Sie die Lizenzbedingungen. OpenOffice ist eine „freie“ Software, die aber den üblichen Restriktionen hinsichtlich einer kommerziellen Nutzung unterliegt. In diesem Buch wird die Version 2.2 verwendet, andere Versionen sind aber ähnlich. Gleiches gilt auch für StarOffice mit Star Base. Natürlich gibt es stets neuere Versionen, die aber zumindest den hier vorausgesetzten Funktionsumfang beinhalten sollten.

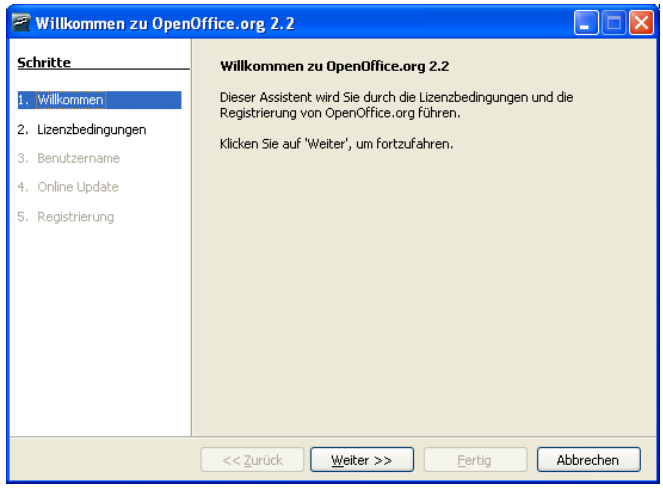
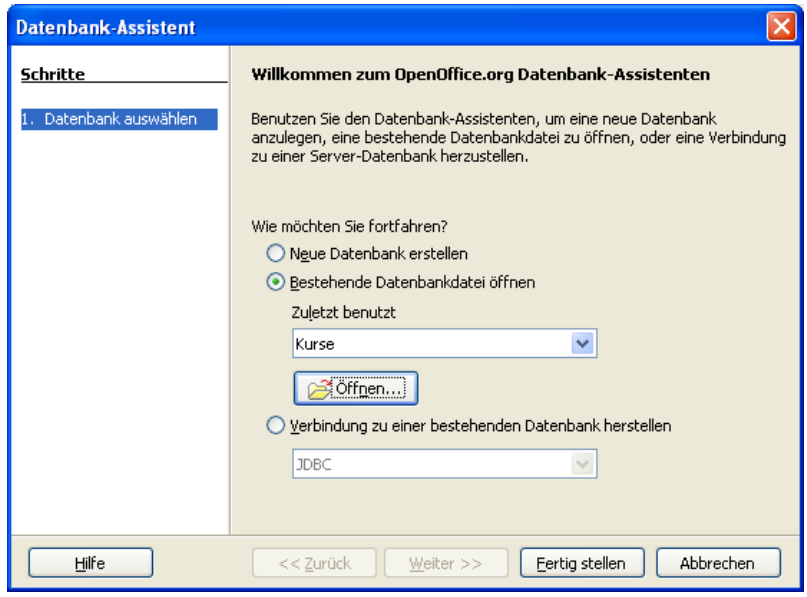


Abbildung 3.50
Start des Installationsassistenten für OpenOffice

Abbildung 3.51
Installation von OpenOffice

Nach der Installation von OpenOffice können Sie openBase aufrufen. Suchen Sie zuvor auf der CD unter *datenbanken/openBase/kurse* nach der Kursdatenbank *kurse.odb*. Wenn Sie die Datei direkt mit dem Windows-Explorer in ein Verzeichnis Ihres Rechners kopieren, müssen Sie an den Schreibschutz denken. Öffnen Sie die kopierte Datei mit einem rechten Mausklick, wählen Sie EIGENSCHAFTEN und entfernen Sie gegebenenfalls den Haken für den Schreibschutz. Danach können Sie die Datei in openBase mit der Option BESTEHENDE DATENBANKDATEI ÖFFNEN, der Wahl der Datei und der Schaltfläche ÖFFNEN laden.

Abbildung 3.52
Öffnen der Datenbank



Damit steht die Datenbank zur Bearbeitung zur Verfügung. Wählen Sie TABELLEN, um eine Übersicht über alle Tabellen zu bekommen. Dies kann beim ersten Mal einen Augenblick dauern. Sie sehen eine Datenbank mit den Tabellen (siehe Abbildung 3.53). Den Inhalt der Tabellen können Sie durch Auswahl mit einem einfachen Klick rechts als Vorschau sehen. Den kompletten Inhalt einer Tabelle können Sie sich mit einem Doppelklick auf den Tabellennamen oder über die rechte Maustaste mit der Option ÖFFNEN ansehen. Sie sehen den kompletten Tabelleninhalt und können ihn mit dem Inhalt im Anhang vergleichen.

Das Fenster schließen Sie mit DATEI/SCHLIESSEN.

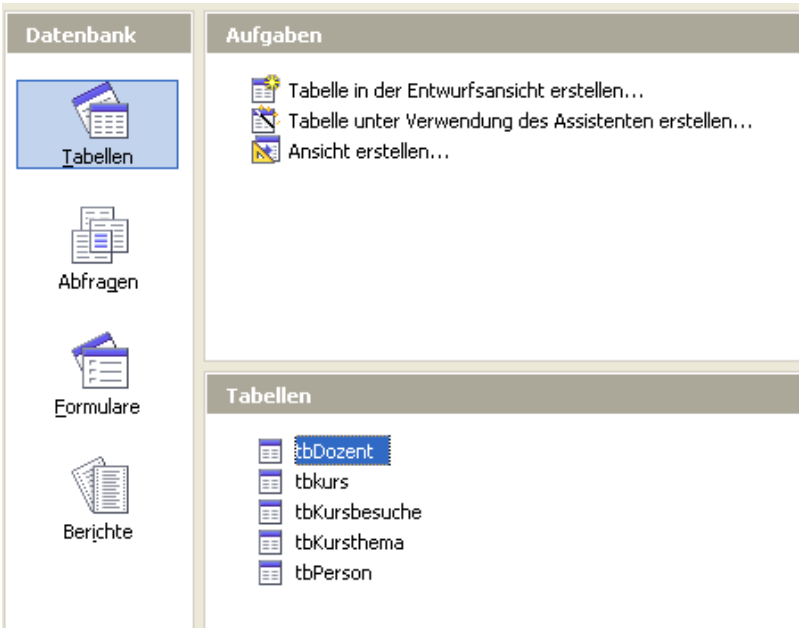


Abbildung 3.53
Übersicht Tabellen in openBase

Beziehungen

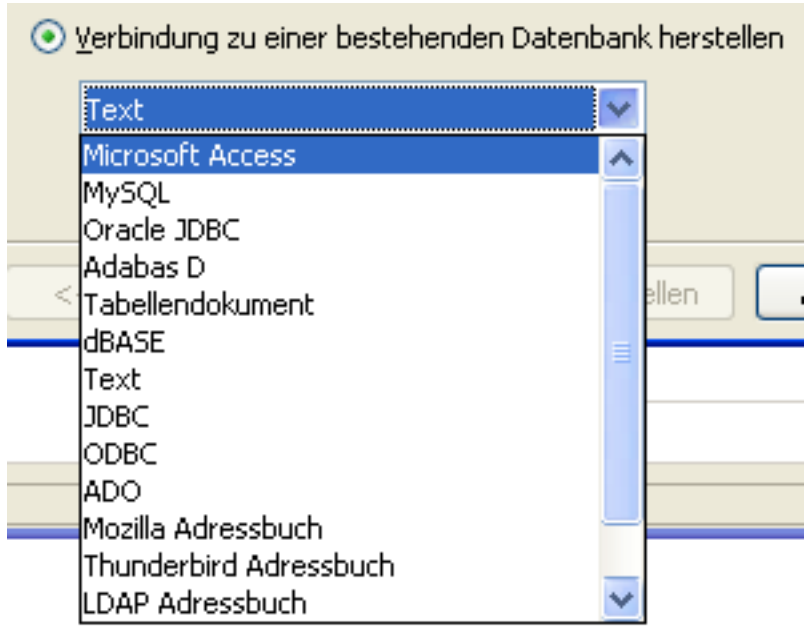
Die Beziehungen zwischen den Tabellen können Sie über EXTRAS/BEZIEHUNGEN sichtbar machen. Sie finden eine ähnliche Darstellung wie in Abbildung 3.1. Durch einen Doppelklick auf eine Beziehung können Sie die beteiligten Felder sowie die beim Ändern oder Löschen von Daten durchzuführenden Aktionen sehen. Die Bedeutung der Beziehungen sehen wir uns in Kapitel 8 an. Sie können dieses Fenster wieder mit DATEI/SCHLIESSEN beenden.

Noch eine Anmerkung für die Nutzer von OpenOffice: openBase kann auch als „Oberfläche“ für andere Datenbankmanagementsysteme verwendet werden. So finden Sie in der Liste möglicher Datenbanken in Abbildung 3.54 auch bekannte Systeme wie MS Access, MySQL und Oracle. Auch die Standardverbindungen ODBC, JDBC und ADO stehen zur Verfügung. Die Nutzung dieser Verbindungen kann openBase zu einer attraktiven Oberfläche und zu einer Alternative zu den hier vorgestellten Oberflächen für die ande-

ren Datenbanken machen. Wegen des teilweise untypischen Verhaltens der Oberfläche und Irritationen bei dem zu verwendenden SQL soll dies hier aber nicht geschehen.

Wenn Sie sich mit openBase und dem anderen Datenbanksystem aber schon ein wenig auskennen, können Sie diesen Weg aber probeweise nutzen.

Abbildung 3.54
Verbindung aus OpenOffice zu anderen Datenbanksystemen



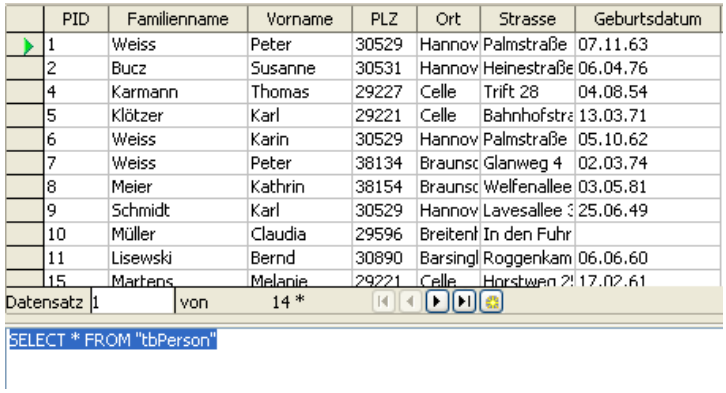
Abschließend soll eine erste Abfrage in openBase erstellt werden, um den Mechanismus zu verstehen. Wählen Sie dazu links ABFRAGEN. Sie haben nun unter AUFGABEN die Möglichkeit, eine Abfrage in der Entwurfsansicht mithilfe eines Assistenten oder in der SQL-Ansicht zu erstellen.

Der Assistent soll uns hier nicht interessieren. Die Entwurfsansicht entspricht der grafischen Oberfläche, die SQL-Ansicht einem reinen Eingabefenster. Wählen Sie die SQL-Ansicht. Sie erhalten ein leeres Fenster. Geben Sie ein:

```
SELECT * FROM tbPerson;
```

Führen Sie die Abfrage anschließend aus, indem Sie entweder das Symbol mit dem grünen Haken oder den Menübefehl BEARBEITEN/ABFRAGE AUSFÜHREN auswählen. Sie sollten eine Tabelle wie in Abbildung 3.55 erhalten.

Sie können dieselbe Abfrage auch in der grafischen Oberfläche von openBase darstellen. Dafür müssen Sie entweder die Schaltfläche mit dem gelben Konstruktionsdreieck oder im Menü ANSICHT/DESIGN-ANSICHT AN-/AUSSCHALTEN auswählen. Das Ergebnis ist die SQL-Anweisung in der Design-Ansicht und sollte etwa wie in Abbildung 3.56 aussehen.

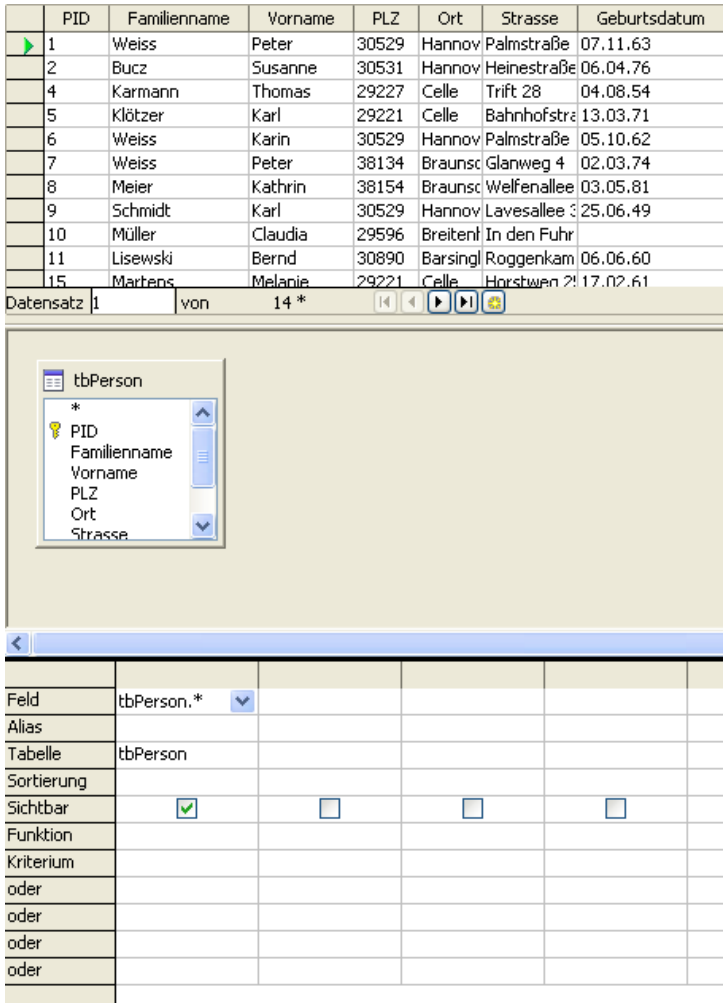


	PID	Familienname	Vorname	PLZ	Ort	Strasse	Geburtsdatum
▶	1	Weiss	Peter	30529	Hannov	Palmstraße	07.11.63
	2	Bucz	Susanne	30531	Hannov	Heinestraße	06.04.76
	4	Karmann	Thomas	29227	Celle	Trift 28	04.08.54
	5	Klötzer	Karl	29221	Celle	Bahnhofstra	13.03.71
	6	Weiss	Karin	30529	Hannov	Palmstraße	05.10.62
	7	Weiss	Peter	38134	Braunsc	Glanweg 4	02.03.74
	8	Meier	Kathrin	38154	Braunsc	Welfenallee	03.05.81
	9	Schmidt	Karl	30529	Hannov	Lavesallee 3	25.06.49
	10	Müller	Claudia	29596	Breitenl	In den Fuhr	
	11	Lisewski	Bernd	30890	Barsingl	Roggenkam	06.06.60
	15	Martens	Melanie	29221	Celle	Horstwen 2	17.02.61

Datensatz 1 von 14 *

```
SELECT * FROM "tbPerson"
```

Abbildung 3.55
Ergebnis der Abfrage in openBase



	PID	Familienname	Vorname	PLZ	Ort	Strasse	Geburtsdatum
▶	1	Weiss	Peter	30529	Hannov	Palmstraße	07.11.63
	2	Bucz	Susanne	30531	Hannov	Heinestraße	06.04.76
	4	Karmann	Thomas	29227	Celle	Trift 28	04.08.54
	5	Klötzer	Karl	29221	Celle	Bahnhofstra	13.03.71
	6	Weiss	Karin	30529	Hannov	Palmstraße	05.10.62
	7	Weiss	Peter	38134	Braunsc	Glanweg 4	02.03.74
	8	Meier	Kathrin	38154	Braunsc	Welfenallee	03.05.81
	9	Schmidt	Karl	30529	Hannov	Lavesallee 3	25.06.49
	10	Müller	Claudia	29596	Breitenl	In den Fuhr	
	11	Lisewski	Bernd	30890	Barsingl	Roggenkam	06.06.60
	15	Martens	Melanie	29221	Celle	Horstwen 2	17.02.61

Datensatz 1 von 14 *

Feld	tbPerson.*				
Alias					
Tabelle	tbPerson				
Sortierung					
Sichtbar	☑	☐	☐	☐	
Funktion					
Kriterium					
oder					
oder					
oder					
oder					

Abbildung 3.56
Design-Ansicht in openBase

Das Umschalten zwischen der Design-Ansicht und der SQL-Ansicht funktioniert zwar theoretisch in beiden Richtungen, tatsächlich gibt es aber Probleme. Einerseits werden wir später SQL-Anweisungen verwenden, die teilweise für die Design-Ansicht zu komplex sind und dort nicht dargestellt werden können. Andererseits wird bei der Umschaltung von der Design-Ansicht in die SQL-Ansicht die SQL-Anweisung komplett neu generiert. Das kann zu einer anderen Darstellung führen, da der Generator bestimmte Dinge rein schematisch durchführt. Wenn Sie beispielsweise jetzt wieder auf die SQL-Ansicht zurückschalten, indem Sie dieselbe Schaltfläche wieder betätigen, steht als SQL-Anweisung jetzt

```
SELECT * FROM "tbPerson"
```

dort. Die eingefügten Anführungszeichen sind zwar harmlos und insofern sogar sinnvoll, dass sie in jedem Fall zu einer korrekten Schreibweise des Namens ohne Umsetzung durch den SQL-Interpreter führen, zeigen aber andererseits, dass hier die SQL-Anweisung neu generiert wurde. Dies kann später auch zu sehr unübersichtlichen Anweisungen führen, wenn insbesondere Klammern in größerer Zahl entstehen.

Weitere Hinweise zur Nutzung von openBase finden Sie in Anhang A.

4

Mit SQL Daten abfragen (SELECT)

Datenbanken dienen der Verwaltung von Informationen. Die wichtigste Aufgabe von SQL besteht also zunächst darin, Informationen aus relationalen Datenbanken zu gewinnen. SQL bietet hierfür die SELECT-Anweisung. Daher ist die SELECT-Anweisung die in der Praxis meistverwendete und wichtigste Anweisung überhaupt.

Jede SELECT-Anweisung liefert eine Menge von Datensätzen (Tupeln) derselben Art, also mit festgelegten Datenfeldern (Spalten) aus einer oder mehreren Tabellen. Um die richtigen Felder und die richtigen Datensätze aus den entsprechenden Tabellen zu ermitteln, muss die SELECT-Anweisung entsprechend formuliert werden. Dabei zeigt sich, dass die SELECT-Anweisung zugleich die einfachste und komplizierteste Anweisung ist. Einfach ist sie, da sie nur aus wenigen, klar strukturierten Bausteinen besteht. Schwierig ist sie, weil diese Bausteine in vielerlei Art kombiniert werden können. Schwierig ist sie auch aufgrund der Performance-Überlegungen, die sie in komplexen Datenbanken mit einer großen Anzahl von Tabellen und Datensätzen erfordert.

4.1 SELECT – die Syntax

Wir wollen die einzelnen Bausteine schrittweise erarbeiten. Damit Sie aber bereits wissen, wohin unsere Reise führt, beginnen wir mit der (weitgehend vollständigen) Syntax der SELECT-Anweisung.

Sie lautet:

SELECT-Syntax

SELECT [DISTINCT|ALL] *ausdrucksliste*

FROM *tabelle [joinliste]*

[WHERE *bedingungsliste]*

[GROUP BY *ausdrucksliste]*

[HAVING *bedingungsliste]*

[ORDER BY *ausdrucksliste [ASC|DESC]];*

Unter einer Ausdrucksliste können Sie sich hier zunächst eine Liste von Datenfeldern vorstellen. Ausdrücke können neben Feldern auch aus Berechnungen, Funktionen und verschiedenen Kombinationen dieser Elemente bestehen. Darauf werden wir in Kapitel 5 ausführlich eingehen. Hier reicht es uns also, uns zunächst eine Liste von Datenfeldern aus einer oder mehreren Tabellen vorzustellen.

Auf Deutsch bedeutet das dann also etwa: „Wähle die benötigten Felder aus den Tabellen aus, beachte einige Einschränkungen, bilde bei Bedarf Pakete (Gruppen) mit weiteren Einschränkungen und sortiere das Ganze abschließend."

> **SELECT = Datensatzmenge**
>
> Eine SELECT-Anweisung liefert immer eine Datensatzmenge. Das Ergebnis können Sie sich also immer als „virtuelle Tabelle" vorstellen. Genau wie richtige Tabellen hat diese virtuelle Tabelle Felder als Spalten und Datensätze als Zeilen.

Übersicht

Sie sehen, dass viele optionale Teile (in den [eckigen] Klammern) stehen, die zunächst entfallen können. Trotzdem sollen die einzelnen Bestandteile der Übersicht wegen hier bereits kurz erläutert werden.

Die geschweiften Klammern weisen auf Wiederholungsgruppen hin, beinhalten also Elemente, die mehrfach auftreten können.

- Die erste Zeile mit `SELECT ausdrucksliste` bestimmt, welche Felder der ermittelten Datensätze angezeigt werden sollen, sie legt also die Spalten der Datensätze in der Ergebnismenge und somit deren Struktur fest.
- Mit `FROM` werden alle Tabellen angegeben, auf die in der gesamten SELECT-Anweisung zurückgegriffen wird, sowie deren Verbindungen untereinander beschrieben (`JOIN`). Im einfachsten Fall werden alle Datensätze einer Tabelle ermittelt und bilden die Zeilen der Ergebnistabelle. Somit ist mit `SELECT ausdrucksliste FROM tabelle` eine Ergebnistabelle bereits vollständig definiert. Das Ergebnis kann ermittelt und angezeigt werden. Alle weiteren Anweisungen modifizieren dieses Ergebnis „nur" noch.
- Die `WHERE`-Klausel beschreibt die Bedingungen, die ein Datensatz erfüllen muss (in Form seiner Werte), um in die Ergebnismenge, die die SELECT-Anweisung liefert, aufgenommen zu werden. Hier werden also die ermittelten Ergebniszeilen eingeschränkt.
- Die `GROUP`-Klausel erlaubt es, Gruppen von Datensätzen zu einem Gruppendatensatz zusammenzufassen.
- Die `HAVING`-Klausel gibt wiederum Bedingungen für diese Gruppendatensätzen an, die erfüllt sein müssen, um einen gruppierten Datensatz in die Ergebnismenge der SELECT-Anweisung aufzunehmen. Hier werden also die ermittelten Gruppendatensätze eingeschränkt.
- Mit der `ORDER BY`-Klausel können die Datensätze und die Gruppendatensätze vor ihrer Ausgabe abschließend sortiert werden.

Wichtig ist in jedem Fall, die Reihenfolge der einzelnen Klauseln beizubehalten. Eine WHERE-Klausel steht immer vor der GROUP-Klausel. Eine ORDER BY-Klausel kommt niemals vor einem HAVING und Entsprechendes gilt für die anderen Klauseln.

Reihenfolge beachten

Abgeschlossen wird eine SELECT-Anweisung – wie jede SQL-Anweisung – durch ein Semikolon (;).

4.2 Einfache Abfragen

Sie sehen, es ist eigentlich ganz einfach. Beschränken Sie sich auf die tatsächlich notwendigen Angaben, so können Sie sehr schnell eine SELECT-Anweisung formulieren, die tatsächlich auch erste Informationen liefert.

Wir wollen damit beginnen, eine SELECT-Anweisung zu formulieren, die alle Personen aus unserer Tabelle tbPerson anzeigt:

Beispiel

```
SELECT Familienname, Vorname
FROM tbPerson;
```

Listing 4.1
Alle Personen mit Namen

Das Ergebnis ist eine Liste, in der in jeder Zeile der Familienname und anschließend der Vorname der Personen angezeigt werden.

Familienname	Vorname
Weiss	Peter
Bucz	Susanne
Karmann	Thomas
Klötzer	Karl
Weiss	Karin
Weiss	Peter
Meier	Kathrin
Schmidt	Karl
Müller	Claudia
Lisewski	Bernd
Martens	Melanie
Schlachter	Dieter
Peredy	Helmut
Ruppert	Nicola
Sander	
Cromberg	Jörg
Schulze	Tanja
Winter	Petra
Plate	Ulrich
Magerkurth	Melissa

Abbildung 4.1
Ergebnis der SELECT-Anweisung

Vertauschen Sie beide Angaben, so erkennen Sie, dass die Reihenfolge der Feldnamen in der Namensliste die Reihenfolge der Werte in der Ergebnismenge direkt bestimmt.

Reihenfolge der Felder

Listing 4.2
Geänderte Reihenfolge

```
SELECT Vorname, Familienname
FROM tbPerson;
```

Abbildung 4.2
Ergebnis bei Vertauschung der beiden Felder

Vorname	Familienname
Peter	Weiss
Susanne	Bucz
Thomas	Karmann
Karl	Klötzer
Karin	Weiss
Peter	Weiss
Kathrin	Meier
Karl	Schmidt
Claudia	Müller
Bernd	Lisewski
Melanie	Martens
Dieter	Schlachter
Helmut	Peredy
Nicola	Ruppert
	Sander
Jörg	Cromberg
Tanja	Schulze
Petra	Winter
Ulrich	Plate
Melissa	Magerkurth

Die Feldnamensliste lässt sich jetzt leicht erweitern. Sollen die Postleitzahl und der Wohnort ergänzt werden, wird die SQL-Anweisung wie folgt erweitert:

Listing 4.3
Komplette Adresse

```
SELECT Vorname, Familienname, PLZ, Ort
FROM tbPerson;
```

Platzhalter *

Möchten Sie alle Spalten einer Tabelle anzeigen, so ergäbe dies eine ziemlich lange Liste, die außerdem bei eventuellen Änderungen der Tabelle wieder zu ergänzen wäre. Dies ist besonders bei `SELECT`-Anweisungen, deren Ergebnis unmittelbar dem Anwender angezeigt werden soll oder die Testzwecken dienen, recht umständlich. Daher gibt es hier die aus Windows gewohnte Kurzform mit `*` als Platzhalter, der für alle Felder steht. Sollen also alle Felder der Tabelle `tbPerson` angezeigt werden, können Sie kurz schreiben:

Listing 4.4
Komplette Tabellenansicht

```
SELECT * FROM tbPerson;
```

Sie erhalten das Ergebnis in Abbildung 4.3.

PID	Familienname	Vorname	PLZ	Ort	Strasse	Geburtsdatum
1	Weiss	Peter	30529	Hannover	Palmstraße 6	07.11.1963
2	Bucz	Susanne	30531	Hannover	Heinestraße 23	06.04.1976
4	Karmann	Thomas	29227	Celle	Trift 28	04.08.1954
5	Klötzer	Karl	29221	Celle	Bahnhofstraße 2	13.03.1971
6	Weiss	Karin	30529	Hannover	Palmstraße 6	05.10.1962
7	Weiss	Peter	38134	Braunschweig	Glanweg 4	02.03.1974
8	Meier	Kathrin	38154	Braunschweig	Welfenallee 23	03.05.1981
9	Schmidt	Karl	30529	Hannover	Lavesallee 32	25.06.1949
10	Müller	Claudia	29596	Breitenhees	In den Fuhren 12	
11	Lisewski	Bernd	30890	Barsinghausen	Roggenkamp 10	06.06.1960
15	Martens	Melanie	29221	Celle	Horstweg 258	17.02.1961
17	Schlachter	Dieter	29227	Celle	Mondhagen 43	02.02.1961
23	Peredy	Helmut	29221	Celle	Mauernstraße 2	23.02.1956
24	Ruppert	Nicola	29301	Bergen	Welfenallee 23	25.02.1962
25	Sander		29223	Celle	Marxallee 12	05.02.1953
26	Cromberg	Jörg	38152	Braunschweig	Nordring 13	07.06.1991
31	Schulze	Tanja	29308	Winsen	Berliner Ring 23	09.11.1992
32	Winter	Petra	29320	Hermannsburg	Immenhoop 51	30.12.1989
34	Plate	Ulrich	30529	Hannover	Gutenberggasse 5	02.12.1986
37	Magerkurth	Melissa	29336	Nienhagen	Am Tümpel 3	04.09.1951

Abbildung 4.3
Alle Felder der Tabelle tbPerson mit SELECT * FROM tbPerson;

Qualifizierung

Solange der Name eines Feldes eindeutig ist, reicht er zur Bezeichnung für die SELECT-Anweisung aus. Innerhalb einer Tabelle müssen alle Namen eindeutig sein. Zwei Felder mit demselben Namen sind verboten. Schwierigkeiten können sich ergeben, wenn mehrere Tabellen in einer SELECT-Anweisung verwendet werden. Sind in mehreren Tabellen Felder gleichen Namens vorhanden, beispielsweise die PID in den Tabellen tbPerson und tbDozent, so ist für den SQL-Interpreter nicht mehr eindeutig erkennbar, welches Feld gemeint ist. In diesen Fällen helfen Sie ihm mit einer sogenannten Qualifizierung. Dabei wird dem Spaltennamen der Name der Tabelle – getrennt durch einen Punkt – vorangestellt.

Für den SQL-Interpreter ist tbPerson.PID eindeutig von tbDozent.PID zu unterscheiden. Nachdem wir im Moment noch mit einer Tabelle arbeiten, ist eine Qualifizierung nicht zwingend erforderlich, aber gewöhnen Sie sich die Schreibweise frühzeitig an, lassen sich später mancherlei Probleme leicht erkennen oder gleich vermeiden.

Info

Bei großen Datenbankanwendungen reicht auch der Tabellenname unter Umständen nicht für eine eindeutige Bezeichnung aus. Daher werden nach demselben Schema weitere Qualifizierer vorangestellt, etwa das Datenbankschema oder der Name der Datenbank, sodass man dann beispielsweise

```
Kurse.tbPerson.PID
```

erhält. Man spricht auch von vollständiger Qualifizierung. Bis auf die Qualifizierung mit dem Tabellennamen wird dies aber im weiteren Verlauf dieses Buches nicht weiter berücksichtigt. Sie sollten dies bei Bedarf in der Dokumentation Ihres Datenbanksystems nachlesen oder die Richtlinien der Datenbankadministratoren erfragen.

Qualifizieren Sie die Feldnamen aus unserer Beispielabfrage mit dem Tabellennamen, erhalten Sie:

Listing 4.5
Qualifizierte Feldnamen

```
SELECT tbPerson.vorname, tbPerson.familienname, tbPerson.PLZ,
       tbPerson.Ort
FROM tbPerson;
```

Dies funktioniert auch mit dem Platzhalter * wunderbar, womit Sie dann alle Felder abfragen können:

Listing 4.6
Qualifikation für alle Feldnamen

```
SELECT tbPerson.*
FROM tbPerson;
```

Die Felder der Ergebnisdatenmenge einer SELECT-Anweisung tragen die gleichen Namen wie die Felder in der Datenbank. Die Namen werden also unmittelbar übernommen und stehen bei einer Ausgabe im Spaltenkopf. Sie können die Namen aber ändern, was sowohl für die Anzeige als auch für die Weiterverarbeitung in einem Programm wichtig sein kann. Fügen Sie dazu in der SELECT-Anweisung an den Feldnamen ein sogenanntes Alias an.

Alias = Ersatznamen

Die Alias sind Ersatznamen, die bei einem Feldnamen (und auch bei einem beliebigen Ausdruck) ergänzt werden können und dann als Ersatzname des Feldes in der Ergebnismenge der SELECT-Anweisung verwendet werden.

So liefert die Ergebnismenge der Anweisung

Listing 4.7
Feldnamen mit Alias

```
SELECT Vorname AS Vorname, Familienname AS Name,
       PLZ AS "Postleitzahl", Ort AS "Wohnort der Person"
FROM tbPerson;
```

eine Ergebnismenge, deren Felder VORNAME, NAME, Postleitzahl und Wohnort der Person sind. Die Groß-/Kleinschreibung wird dabei von verschiedenen SQL-Interpretern konsequent in Großschreibung umgesetzt (siehe Abbildung 4.4).

Abbildung 4.4
Ausschnitt aus der Darstellung mit Alias

VORNAME	NAME	Postleitzahl	Wohnort der Person
Peter	Weiss	30529	Hannover
Susanne	Bucz	30531	Hannover
Thomas	Karmann	29227	Celle
Karl	Klötzer	29221	Celle
Karin	Weiss	30529	Hannover
Peter	Weiss	38134	Braunschweig

Soll eine gemischte Schreibweise realisiert werden, so können dafür die doppelten Anführungsstriche um das Alias gesetzt werden. Sie sehen außerdem, dass über die Verwendung der Anführungsstriche auch Leerzeichen in den Namen eingefügt werden können, was vor allem bei einer direkten Ausgabe für einen Endanwender sinnvoll sein kann.

Leerzeichen und Sonderzeichen

Leerzeichen und andere Sonderzeichen erhöhen aus Anwendersicht oft die Lesbarkeit von Namen. Die Angabe Stundensatz in _ ist besser lesbar als StundensatzInEuro oder Stundensatz in €.

Als Namen für Datenbankfelder sollten Leerzeichen oder andere Sonderzeichen aber vermieden werden. Für den Anwender (aber bitte niemals für weiterverarbeitende Programme) können Sie dann einen Feldnamen `Stundensatz_in_Euro` immer noch mit einem

```
SELECT Stundensatz_in_Euro AS "Stundensatz in _"
FROM tbMitarbeiter;
```

für die Ausgabe mit dem gewünschten Alias versehen, um den Anwender zufriedenzustellen.

Keine Sonderzeichen in Datenbanknamen

Sollen also Datenbankobjekte wie Tabellen und Spalten verarbeitet werden, so ist es im Allgemeinen keine gute Idee, Leerzeichen oder andere „schöne" Symbole und Sonderzeichen als Namen der Tabellen und Datenfelder zu verwenden. Jeder Programmierer oder Datenbankadministrator bekommt bei Feldnamen wie „Stundensatz in EURO", „geschätzter Aufwand" oder „Frist beachten!" sofort mehr als nur etwas steife Nackenhaare.

Regeln für die Namensvergabe

Generell sollten Sie versuchen, ausschließlich Folgendes zu verwenden:

- Buchstaben – denken Sie daran, dass die meisten SQL-Interpreter Groß- und Kleinschreibung nicht unterscheiden, vermeiden Sie auch Umlaute und das „ß".
- Ziffern – aber nicht als erstes Zeichen eines Namens.
- Den Unterstrich „_", wenn es nicht anders geht.
- Keine SQL-Schlüsselwörter wie `SELECT`, `INSERT`, `UPDATE` oder `DELETE`.

CamelCase-Schreibweise

Die sogenannte „CamelCase-Schreibweise", bei der in zusammengesetzten Wörtern jedes neue Wort mit einem Großbuchstaben beginnt, wäre für Datenbankfelder sinnvoll. Der Feldname `StundensatzInEuro` ist grundsätzlich ein vernünftiger Name. Leider gibt es hier aber eine Einschränkung. Die meisten Datenbanksysteme im Großrechnerumfeld wie auch SQL-Interpreter, die aus diesem Umfeld kommen, kennen, wie Sie wissen, nur Großschreibung. Dies ist auch im SQL-Standard so definiert und wird bei unseren Beispielsystemen von MySQL, Oracle und Firebird konsequent umgesetzt. Es ist also konform zum ANSI-SQL-Standard. Im Umfeld von Betriebssystemen, sei es Windows, sei es Linux, wird dagegen bei einigen Datenbanksystemen sorgfältig zwischen Groß- und Kleinschreibung unterschieden. MySQL im Linux-Umfeld (umschaltbar) und noch deutlicher openBase auch im Windows-Umfeld zeigen dies bei unseren Beispielsystemen. Bei MS Access finden wir ein Windows-typisches Verhalten ohne besondere Beachtung von Groß- und Kleinschreibung.

Alias für die Programmierung

Beachtet Sie dies, können Sie also mit obigen Hinweisen und etwas gutem Willen Namen verwenden, die wenig Probleme bereiten. Trotzdem findet man gerade im Umfeld von MS Access, openBase oder anwendernahen Entwicklungen immer wieder kunstvolle Namen mit Leerzeichen oder Sonderzeichen. Sind sie erst einmal in der Datenbank vorhanden, ist es oft schwer, sie wieder zu ändern. Soll eine Datenbank längerfristig Anwendung finden, versuchen Sie trotzdem zunächst diese Namen zu ändern. Ist dies nicht möglich, weil es zu aufwendig ist, so hilft bei SQL-Anweisungen die Verwendung

der Alias-Schreibweise auch in umgekehrter Richtung weiter und sollte konsequent in jeder Anweisung verwendet werden, bevor die Ergebnisse weiterverarbeitet werden. Heißt beispielsweise das Feld in einer Tabelle `tbMitarbeiter` tatsächlich `Stundensatz in Euro`, so behelfen Sie sich in der Abfrage mit:

Listing 4.8 Umsetzung in einen IT-konformen Namen

```
SELECT "Stundensatz in EURO" AS Stundensatz_in_Euro
FROM tbMitarbeiter;
```

Hier wird der Spieß also umgedreht, aus dem unglücklichen Feldnamen wird ein programmierkompatibles Alias. Sie können das Alias natürlich auch zusammenschreiben.

Wie erwähnt hilft die „CamelCase-Schreibweise“ bei vielen SQL-Interpretern im Zusammenhang mit einem Alias auch nicht weiter, da der Interpreter aus `StundensatzInEuro` sofort `STUNDENSATZINEURO` macht, was auch nicht wirklich überzeugt.

Tipp

Generell lautet der Tipp: Sollen zusammengesetzte Wörter in Namen verwendet werden, nutzen Sie den Unterstrich _. Dieser wird von fast jedem SQL-Interpreter und fast jeder relationalen Datenbank akzeptiert.

Soweit die Theorie, aber wir sind bekanntlich in der EDV. Also ist nichts beständiger als die Ausnahme. Vielleicht haben Sie die obigen Anweisungen schon in MS Access oder MySQL ausprobiert und fragen sich, was Sie (oder der Autor dieses Buches) hier wieder falsch gemacht haben. Es könnte zumindest auch daran liegen, dass sowohl MS Access als auch MySQL hier ein vom Standard abweichendes Verhalten zeigen.

Alias in MS Access

Alias in MS Access

MS Access ist – wie erwähnt – stark von Windows geprägt. Daher werden hier die Feldnamen nicht in Großschreibung umgesetzt, sondern unverändert beibehalten. Die Einschließung in doppelte Anführungsstriche funktioniert ebenfalls nur sehr bedingt, diese werden nämlich als Teil des Alias übernommen. MS Access müsste also für die obige `SELECT`-Anweisung etwa das Ergebnis in Abbildung 4.5 geliefert haben.

Abbildung 4.5 Ergebnis in MS Access bei SQL-standardkonformer Alias-Angabe

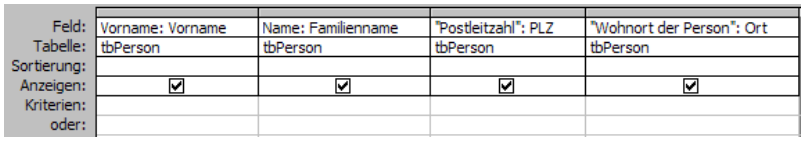
Vorname	Name	"Postleitzahl"	" Wohnort der Person"
Peter	Weiss	30529	Hannover
Susanne	Bucz	30531	Hannover
Thomas	Karmann	29227	Celle
Karl	Klötzer	29221	Celle
Karin	Weiss	30529	Hannover

Um in MS Access sinnvoll mit Ersatznamen arbeiten zu können, müssen die Alias-Angaben in die für MS Access typischen eckigen Klammern `[...]` gesetzt werden. Soll das Standardverhalten der Umsetzung in Großbuchstaben erfolgen, ist dies ebenfalls vorher zu berücksichtigen. In MS Access müsste die `SELECT`-Anweisung also etwa wie folgt lauten:

```
SELECT tbPerson.Vorname AS VORNAME, tbPerson.Familienname AS NAME,
       tbPerson.PLZ AS Postleitzahl,
       tbPerson.Ort AS [Wohnort der Person]
FROM tbPerson;
```

Listing 4.9
Alias in MS Access

Die Angabe der eckigen Klammern ist optional, solange keine Leerzeichen oder sonstigen Sonderzeichen auftreten. In der grafischen Oberfläche von MS Access könnte das dann wie in Abbildung 4.6 aussehen.



Feld:	Vorname: Vorname	Name: Familienname	"Postleitzahl": PLZ	"Wohnort der Person": Ort
Tabelle:	tbPerson	tbPerson	tbPerson	tbPerson
Sortierung:				
Anzeigen:	☑	☑	☑	☑
Kriterien:				
oder:				

Abbildung 4.6
SELECT-Anweisung mit Alias in MS Access

Alias in MySQL

Alias in MySQL

An dieser Stelle sind jetzt noch ein paar Hinweise für die MySQL-Nutzer angebracht. Wie immer unterscheidet der SQL-Interpreter von MySQL die Groß-/Kleinschreibung entsprechend der Konfiguration und dem Betriebssystem auf dem MySQL läuft. Insbesondere kann es zwischen Windows und Linux deutliche Unterschiede geben. Folglich muss das Gesagte unter diesem Blickwinkel betrachtet werden. In den meisten Fällen sollten Sie aber ein standardkonformes Ergebnis, wie in Abbildung 4.7 gezeigt, erhalten.

Vorname	Name	Postleitzahl	Wohnort der Person
Peter	Weiss	30529	Hannover
Susanne	Bucz	30531	Hannover
Thomas	Karmann	29227	Celle
Karl	Klötzer	29221	Celle
Karin	Weiss	30529	Hannover

Abbildung 4.7
Ergebnis der SELECT-Anweisung mit Alias in MySQL

Sie können dies auch mit der Kommandozeilenoberfläche von MySQL testen. Starten Sie *mysql.exe*. Bei Eingabe der SQL-Anweisung am Prompt erhalten Sie das in Abbildung 4.8 gezeigte Ergebnis.

```
mysql> select Vorname AS Vorname, Familienname AS Name, PLZ AS "Postleitzahl", O
rt AS "Wohnort der Person" FROM tbPerson;
+---------+-------------+--------------+--------------------+
| Vorname | Name        | Postleitzahl | Wohnort der Person |
+---------+-------------+--------------+--------------------+
| Peter   | Weiss       | 30529        | Hannover           |
| Susanne | Bucz        | 30531        | Hannover           |
| Thomas  | Karmann     | 29227        | Celle              |
| Karl    | Klötzer     | 29221        | Celle              |
| Karin   | Weiss       | 30529        | Hannover           |
+---------+-------------+--------------+--------------------+
```

Abbildung 4.8
Ergebnis der SELECT-Anweisung mit Alias in mysql.exe

Alias für Tabellen

Wie die Feldnamen können auch die Tabellennamen mit einem Alias versehen werden. Damit kann das Alias des Tabellennamens an allen Stellen der SELECT-Anweisung verwendet werden, an denen sonst der komplette Tabellenname stehen müsste. Insbesondere kann das Alias zur Qualifizierung der Spaltennamen mit dem Tabellennamen verwendet werden.

Tipp

Verwenden Sie das Alias für die Tabellennamen, wann immer möglich. Statt

```
SELECT tbPerson.vorname, tbPerson.familienname, tbPerson.PLZ,
       tbPerson.Ort
FROM tbPerson;
```

können Sie auch einfach schreiben:

```
SELECT p.vorname, p.familienname, p.PLZ, p.Ort
FROM tbPerson p;
```

Das macht die Anweisung kürzer, ist genauso verständlich und erlaubt, das Alias der Tabelle später auch noch an deren Stelle in der SELECT-Anweisung zu nutzen.

Alias in Oracle

Alias in Oracle

Oracle verhält sich hier standardkonform. Namen werden grundsätzlich in Großbuchstaben umgesetzt, sofern sie nicht in Anführungsstrichen stehen.

Alias in openBase

Alias in openBase

Das Datenbankmanagementsystem openBase verhält sich hier im Gegensatz zu MS Access weitgehend standardkonform. Es erfolgt allerdings ebenfalls keine Umsetzung der Alias in Großbuchstaben, was unter Windows den Erwartungen entspricht. Beachten Sie, dass openBase bei der Generierung von SQL-Anweisungen aus der Entwurfssicht die Namen der Datenbankobjekte generell in Anführungszeichen setzt. Außerdem gibt es manchmal Schwierigkeiten, wenn Sie mehrere Alias verwenden.

ALL und DISTINCT

Sicher ist Ihnen schon bei der SELECT-Syntax am Anfang dieses Kapitels die Angabe [DISTINCT|ALL] unmittelbar nach dem SELECT aufgefallen. Zur Erinnerung, die eckigen Klammern bedeuten, dass es sich um optionale Angaben handelt, der Inhalt also entfallen kann. Der senkrechte Strich trennt verschiedene Auswahlmöglichkeiten im Sinne eines „entweder ... oder ...“.

Wird kein Wert angegeben, also

```
SELECT vorname, familienname
FROM tbPerson;
```

geschrieben, ist das gleichbedeutend mit der Angabe des ALL. ALL ist der Standard- oder Default-Wert:

```
SELECT ALL vorname, familienname
FROM tbPerson;
```

Dabei kann es vorkommen, dass die Ergebnismenge mehrere identische Datensätze enthält, also Datensätze, bei denen alle Felder den gleichen Wert aufweisen. In unserem Beispiel wäre das bei dem Herrn Peter Weiß der Fall (siehe Abbildung 4.1). Sollen diese Duplikate unterdrückt werden, kann dies mit der Angabe DISTINCT geschehen.

```
SELECT DISTINCT
   vorname,familienname
FROM tbPerson;
```

Listing 4.10
Elimination von Duplikaten mit DISTINCT

Hier wird die Ergebnismenge vor der Ausgabe auf Duplikate geprüft und diese werden eliminiert.

Info

- Oracle verwendet `UNIQUE` synonym zu `DISTINCT`.
- Einige Datenbankmanagementsysteme erlauben es, das `AS` vor dem Alias wegzulassen, also `SELECT Familienname Name`statt `SELECT Familienname AS Name` zu schreiben. Diese Schreibweise wird hier nicht verwendet, da sie
- weder von allen Systemen unterstützt wird,
- noch der Übersichtlichkeit dient,
- noch Einheitlichkeit bezüglich Alias für Felder und Alias für Tabellen existiert.
- openBase bietet sowohl für `ALIAS` als auch für `DISTINCT` eine Unterstützung in der Entwurfsansicht (siehe Anhang A.5).

Zusammenfassung

Sie haben jetzt die grundlegenden Elemente für eine Datenbankabfrage mit einer `SELECT`-Anweisung kennengelernt. Sie können die gewünschten Felder einschließlich der Tabelle angeben und mit der Syntax `tabelle.feldname` qualifizieren. Sie können Felder und Tabellen mit einem Alias versehen. Sie kennen die Schreibweise `feldname AS "Ersatzname"` für ein Alias. Damit haben wir jetzt die Syntax einer `SELECT`-Anweisung bis hierher zusammengestellt:

Syntax Alias

SELECT [DISTINCT|ALL] *feldname AS alias* { ,*feldname [AS alias]*}

FROM *tabelle [AS alias];*

Sie wissen, dass man den `*` stellvertretend für alle Felder angeben kann. Durch den Zusatz von `DISTINCT` können Sie Duplikate in der Ergebnismenge verhindern.

Übungen zum einfachen SELECT mit und ohne Alias

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Ermitteln Sie aus der Tabelle `tbKursthema` alle Felder für alle Themen. (Ü4.2.1)
2. Ermitteln Sie aus der Tabelle `tbKursthema` das Thema, die Kursbeschreibung und die geplante Dauer des Kurses. Verwenden Sie für den Tabellennamen das Alias `kt` und qualifizieren Sie die Feldnamen mit dem Alias. (Ü4.2.2)
3. Ermitteln Sie aus der Tabelle `tbKursthema` die Kursthemen-Identifikationsnummer (`KTHID`) unter dem Alias `KThID`, also mit kleinem „h“, das eigentliche Kursthema unter dem Ersatznamen `KURSTHEMA` und die Dauer unter dem Alias `"geplante Kursdauer"`. (Ü4.2.3)
4. Ermitteln Sie alle in der Tabelle `tbKursthema` vorkommenden verschiedenen Kursdauern (ohne weitere Felder). (Ü4.2.4)

4.3 Daten sortieren mit der ORDER BY-Klausel

Wie der Ausdruck ORDER BY nahelegt, geht es um die Sortierung der Ergebnismenge. Relationale Datenbanken arbeiten mit Mengen. Mengen sind aber per Definition unsortiert. Das bedeutet, dass die Reihenfolge der Datensätze zufällig ist und allein im technischen Ermessen des Datenbankmanagementsystems liegt. Natürlich wird man bei einem SELECT beobachten, dass die Daten oft in der Reihenfolge ihrer Eingabe erscheinen, aber erstens kennt der eine oder andere Anwender diese Reihenfolge nicht unbedingt und zweitens gibt es dafür keinerlei Garantie. Spätestens, wenn mehrere Tabellen an einem SELECT beteiligt sind, wird die Lage unübersichtlich.

ORDER BY

In vielen Fällen möchten Sie sicher sein, dass die Daten in einer bestimmten Reihenfolge sortiert dargestellt werden. Daher bietet die SELECT-Anweisung mit der ORDER BY-Klausel die Möglichkeit, die Datensätze nach eigenen Kriterien zu sortieren. Dies betrifft natürlich immer nur die Ergebnismenge des SELECT, niemals die tatsächliche Speicherung der Daten in den Tabellen.

Beispiel

Sollen beispielsweise unsere Personen alphabetisch sortiert werden, so ist dazu zunächst der Familienname relevant.

Listing 4.11 Alphabetische Sortierung nach dem Familiennamen

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort
FROM tbPerson p
ORDER BY p.Familienname;
```

Das Ergebnis ist eine Liste der Familiennamen, Vornamen, Postleitzahlen und Orte aufsteigend sortiert nach Familiennamen.

Soll zusätzlich innerhalb der Namen nach Vornamen sortiert werden, kann eine weitere Spalte in die ORDER BY-Klausel aufgenommen werden:

Listing 4.12 Alphabetische Sortierung nach Familienname und Vorname

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort
FROM tbPerson p
ORDER BY p.Familienname, p.Vorname;
```

Das Ergebnis der SELECT-Anweisung ist in Abbildung 4.9 zu sehen. Zunächst erkennen Sie die alphabetische Sortierung der Familiennamen. Die Sortierung innerhalb der Vornamen erkennen Sie beim Familiennamen „Weiss“ an der Anordnung der Datensätze nach dem Vornamen. Natürlich kann auch die erste SELECT-Anweisung, die nur nach dem Familiennamen sortiert, zufällig dasselbe Ergebnis liefern. Die Reihenfolge ist dann aber zufällig und nicht garantiert. Entsprechend ist in Abbildung 4.9 das Ergebnis hinsichtlich der Reihenfolge der beiden Datensätze für „Weiss, Peter“ auch noch zufällig. Hier müssten dann gegebenenfalls weitere Sortierkriterien vorgegeben werden.

Familienname	Vorname	PLZ	Name
Bucz	Susanne	30531	Hannover
Cromberg	Jörg	38152	Braunschweig
Karmann	Thomas	29227	Celle
Klötzer	Karl	29221	Celle
Lisewski	Bernd	30890	Barsinghausen
Magerkurth	Melissa	29336	Nienhagen
Martens	Melanie	29221	Celle
Meier	Kathrin	38154	Braunschweig
Müller	Claudia	29596	Breitenhees
Peredy	Helmut	29221	Celle
Plate	Ulrich	30529	Hannover
Ruppert	Nicola	29301	Bergen
Sander	NULL	29223	Celle
Schlachter	Dieter	29227	Celle
Schmidt	Karl	30529	Hannover
Schulze	Tanja	29308	Winsen
Weiss	Karin	30529	Hannover
Weiss	Peter	38134	Braunschweig
Weiss	Peter	30529	Hannover
Winter	Petra	29320	Hermannsburg

Abbildung 4.9
Ergebnis der SELECT-Anweisung

ASC|DESC

Die Reihenfolge der Sortierung wird dabei stets durch die Reihenfolge der Felder (Ausdrücke) in der `ORDER BY`-Klausel bestimmt und von links nach rechts abgearbeitet. Sortierungen können prinzipiell aufsteigend oder absteigend erfolgen. Dafür bietet SQL die Schlüsselwörter `ASCENDING` (aufsteigend) und `DESCENDING` (absteigend) an, die zumeist mit `ASC` und `DESC` abgekürzt werden. Nicht alle Datenbanken unterstützen die Langform. Der Standardwert ist `ASC`.

Somit ist die obige Abfrage gleichbedeutend mit:

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort
FROM tbPerson p
ORDER BY p.Familienname ASC, p.Vorname ASC;
```

Listing 4.13
Alphabetisch aufsteigende Sortierung

Soll eine Liste absteigend nach dem Geburtsdatum und innerhalb der Geburtsdaten aufsteigend nach Namen sortiert werden, ergäbe sich:

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort, p.Geburtsdatum
FROM tbPerson p
ORDER BY p.Geburtsdatum DESC, p.Familienname ASC, p.Vorname ASC;
```

Listing 4.14
Gemischte absteigende und aufsteigende Sortierungen

Sortierung und Datentyp

Die Sortierreihenfolge hängt von dem Datentyp eines Datenfeldes ab. So gilt für numerische Typen einschließlich der Prozentangaben und Währungsformate die gewohnte Reihenfolge der Zahlenmengen. Bei aufsteigender Sortierung werden kleinere Zahlen nach vorn, größere Zahlen nach hinten sortiert. Entsprechend wird bei Datums- und Zeitangaben vom früheren Datum beziehungsweise der früheren Zeit aufsteigend zu neueren Datums- respektive neueren Zeitangaben sortiert.

Bei alphanumerischen Angaben liegen die Dinge komplizierter. Hier bestimmt der zugrunde liegende ASCII- oder ANSI-Code und die entsprechende `COLLATE`-Angabe zur Auswahl des Zeichensatzes die Reihenfolge der Sortierung. Betrachten Sie dazu das Ergebnis der Testabfrage in Listing 4.15 auf einer fiktiven Tabelle `tbTest`.

Listing 4.15
Fiktive Abfrage zur Sortierreihenfolge bei alphanumerischen Werten

```
SELECT Test
FROM tbTest
ORDER BY Test ASC;
```

Das Ergebnis könnte wie in Abbildung 4.10 dargestellt aussehen.

Abbildung 4.10
Beispiel für die Sortierreihenfolge alphanumerischer Werte

Test
Hannover
_Hannover
19
190
20
21
hannover
Hannover

Es wird stets zunächst das erste Zeichen der Inhalte verglichen, sind diese gleich, das zweite Zeichen, sind diese gleich, das dritte Zeichen bis zum Ende der Zeichenketten. Sind beide Zeichenketten bis zum Ende gleich, wird die kürzere vor die längere Kette sortiert, also beispielsweise „19“ vor „190“.

Bei der Sortierung ist der verwendete Zeichensatz entscheidend. Sie sehen an den beiden letzten Angaben, dass Kleinbuchstaben vor Großbuchstaben liegen. Ziffern wiederum liegen vor Buchstaben. Sonderzeichen wie der Unterstrich oder das Leerzeichen, das im obersten Datensatz vor „Hannover“ steht, liegen wiederum abhängig vom verwendeten Datensatz an anderer Stelle, im Beispiel vor den Ziffern. Sie müssen also bei alphanumerischen Angaben auf den Zeichensatz achten.

Besonders kritisch kann es werden, wenn eigentlich numerische Angaben als alphanumerischer Text gespeichert werden. Sie sehen, dass 19, 20 und 21 richtig angeordnet werden, die 190 aber an der „falschen“ Stelle steht. Dies liegt an dem beschriebenen Mechanismus. Bereits beim Vergleich des ersten Zeichens wird die „20“ hinter die „190“ eingeordnet, da die „2“ größer als die „1“ ist. Die Sortierung von Zahlen in alphanumerischen Feldern funktioniert nur, wenn die Angaben gleich lang sind, wie beispielsweise bei Postleitzahlen, und führende Nullen verwendet werden.

Tipp

Die Speicherung numerischer Angaben in alphanumerischen Feldern sollte also nur in besonders begründeten Fällen erfolgen, da sonst die Sortierung erschwert wird. Ist dies nicht zu vermeiden, können hier eventuell datenbankspezifische Funktionen Abhilfe schaffen, auf die in Kapitel 5 noch einzugehen ist.

Feldname oder Position

Bei der Sortierung haben Sie die Wahl, ob Sie in der `ORDER BY`-Klausel den Namen des Datenfeldes, das Alias oder auch die Nummer des Datenfeldes im `SELECT` angeben. So können Sie statt

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort
FROM tbPerson p
ORDER BY p.Familienname ASC, p.Vorname ASC;
```

auch schreiben

Listing 4.16
Sortierung mit Feldpositionen

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort
FROM tbPerson p
ORDER BY 1 ASC, 2 ASC;
```

Entsprechend der Position des Familiennamens als erstem und des Vornamens als zweitem Datenfeld in der Abfrage erfolgt die Sortierung wie bei der oberen Anweisung.

Außerdem lassen einige Systeme wie Oracle Zusätze zu, wie bei einer Sortierung mit fehlenden Werten in den Datensätzen umgegangen werden soll (siehe Listing 4.17).

Listing 4.17
Berücksichtigung leerer Felder bei der Sortierung

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort
FROM tbPerson p
ORDER BY 1 ASC, 2 ASC NULLS LAST;
```

Die Angabe `NULLS LAST` bewirkt beispielsweise in Oracle, dass alle Datensätze mit fehlenden Vornamen innerhalb eines Familiennamens an das Ende sortiert werden.

Zusammenfassung

Sie haben gesehen, wie der `SELECT`-Anweisung eine Sortierung hinzugefügt werden kann. Bisher haben wir daher den folgenden Umfang der `SELECT`-Anweisung erarbeitet:

SELECT [DISTINCT|ALL] *feldname [AS alias]* { *,feldname [AS alias]*}

FROM *tabelle [AS alias]*

[ORDER BY *feldname [ASC|DESC]* { *,feldname [ASC|DESC]*}*];*

Sie wissen, dass man das Ergebnis jeder `SELECT`-Anweisung aufsteigend (Standard) oder absteigend nach einem oder mehreren Datenfeldnamen sortieren kann. Dabei wird die Reihenfolge der Sortierung durch die Reihenfolge der Feldnamen in der `SELECT`-Anweisung von links nach rechts bestimmt. Je weiter links ein Feld steht, desto entscheidender ist es für die Sortierung. Weiter rechts stehende Felder werden nur bei Gleichheit der linken Felder berücksichtigt.

Sortierung und Performance

Die ORDER BY-Klausel hat nicht unbeträchtliche Auswirkungen auf die Performance. Bei großen Datenmengen wirken sich zwei Faktoren negativ aus. Zum einen ist die Sortierung selbst eine aufwendige Operation. Zum anderen bieten viele Datenbanken an, zunächst eine begrenzte Anzahl von Datensätzen als Ergebnis der Auswertungen zu liefern, beispielsweise die ersten 100 gefundenen Datensätze. Dies geht unter Umständen sehr schnell, wenn keine Sortierung vorgenommen werden soll, da dann einfach die ersten 100 Datensätze gelesen werden. Soll das Ergebnis aber sortiert werden, müssen alle Datensätze zunächst gelesen werden, um festzustellen, welches die 100 ersten Datensätze gemäß der gewünschten Sortierreihenfolge sind. Das kann einen erheblichen Unterschied bei den Antwortzeiten bedeuten.

Müssen Sie daher mit großen Datenmengen arbeiten, sollten Sie bei der Entwicklung und dem Test der SQL-Anweisung wenn möglich zunächst auf die Sortierung verzichten. Entsprechend gilt später auch für produktive SQL-Anweisungen, dass bei einer Sortierung großer Datenbestände zusätzliche Performance-Überlegungen notwendig sind.

Übungen

SELECT-Übungen zur ORDER BY-Klausel

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung:

1. Erstellen Sie auf Basis der Tabelle `tbPerson` eine Liste aller Personen mit `Familienname`, `Vorname` und `Geburtsdatum`, bei der die jüngsten Personen oben stehen. (Ü4.3.1)
2. Ermitteln Sie aus der Tabelle `tbPerson` eine Liste mit beiden Namen, Postleitzahl, Ort und Straße, die nach Orten und innerhalb der Orte nach Postleitzahlen jeweils aufsteigend sortiert ist. Qualifizieren Sie die Felder mit dem Tabellennamen als Alias `p`. (Ü4.3.2)
3. Ermitteln Sie aus der Tabelle `tbKursthema` alle Kurse mit `Kursthema` und `DauerPlan`, wobei diese so sortiert sein sollen, dass die Kurse mit der längsten geplanten Dauer oben stehen. Die geplante Dauer soll wieder unter dem Alias `"geplante Kursdauer"` aufgelistet werden. Probieren Sie, ob Sie das Alias zur Sortierung verwenden können. (Ü4.3.3)
4. Ermitteln Sie aus der Tabelle `tbKursbesuche` eine Liste aller Kursbesuche (mit allen Datenfeldern), die zunächst aufsteigend nach der `KID` sortiert ist. Innerhalb eines Kurses soll aufsteigend nach Zahlweisen und innerhalb der Zahlweisen so sortiert werden, dass die höchsten Rabatte am Anfang stehen. (Ü4.3.4)

4.4 Die Daten mit der WHERE-Klausel auswählen

Bisher haben wir zwar die Felder einschränken können, die wir als Ergebnis einer SELECT-Anweisung erhalten, aber immer alle Datensätze einer Tabelle von der Datenbank geliefert bekommen. Bei großen Tabellen bedeutet das

nicht nur eine Belastung der Datenleitungen, sondern insbesondere viel zu große und unübersichtliche Mengen. Daher müssen auch die Datensätze, die das Ergebnis einer SELECT-Anweisung sind, meistens eingeschränkt werden. Dies ist die Aufgabe der WHERE-Klausel.

Beispiel

Soll beispielsweise vorbereitend für eine Glückwunschkarte die Liste aller Personen ermittelt werden, die in Celle wohnen und im Februar Geburtstag haben, so könnte dies mit einer SQL-Anweisung wie in Listing 4.18 geschehen.

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort, p.Strasse,
       p.Geburtsdatum
FROM tbPerson p
WHERE ((p.Ort='Celle') AND (p.Geburtsdatum Like '*.02.*'))
ORDER BY p.Familienname, p.Vorname;
```

Listing 4.18
Alle Personen aus Celle, die im Februar Geburtstag haben (MS Access)

MS Access-Beispiel

Das Ergebnis ist in Abbildung 4.11 zu sehen. Es muss auf die Tabelle tbPerson zugegriffen werden. Sinnvoll ist es, neben dem Namen auch den Geburtstag und die Adresse anzuzeigen, die für den Glückwunsch benötigt werden. Eine alphabetische Sortierung ist generell sinnvoll. Zusätzlich werden die Datensätze eingeschränkt. Es sollen nur solche Datensätze erscheinen, bei denen in der Tabelle tbPerson im Feld Ort der Wert „Celle“ steht. Zusätzlich soll der Geburtstag im Februar liegen. Ausgehend von einer Datumsdarstellung im Format „Tag.Monat.Jahr“ (also „tt.mm.jjjj“) wird der Stern als Platzhalter verwendet, um hinsichtlich des Tages und des Jahres keine Einschränkung zu machen.

Familienname	Vorname	PLZ	Ort	Strasse	Geburtsdatum
Martens	Melanie	29221	Celle	Horstweg 258	17.02.1961
Peredy	Helmut	29221	Celle	Mauernstraße 2	23.02.1956
Sander		29223	Celle	Marxallee 12	05.02.1953
Schlachter	Dieter	29227	Celle	Mondhagen 43	02.02.1961

Abbildung 4.11
Ergebnis der SELECT-Anweisung mit WHERE-Klausel in MS Access

MySQL-Beispiel

In diesem ersten Beispiel wird bewusst schon ein Problem angesprochen, das oft im Zusammenhang mit der WHERE-Klausel auftritt: kleine Abweichungen vom Standard, Erweiterungen und sprachspezifische Einstellungen. Das Beispiel funktioniert in dieser Form nur in MS Access. Wollen Sie dasselbe Ergebnis mit MySQL oder einer der anderen Datenbanken mit den dortigen Platzhaltern und dem englischen Datumsformat erreichen, lautet die entsprechende SELECT-Anweisung:

```
SELECT p.Familienname, p.Vorname, p.PLZ, p.Ort, p.Strasse,
       p.Geburtsdatum
FROM tbPerson p
WHERE (p.Ort='Celle') and (p.Geburtsdatum LIKE '%-02-%')
ORDER BY p.Familienname, p.Vorname;
```

Listing 4.19
Alle Personen aus Celle, die im Februar Geburtstag haben

Das Ergebnis ist in Abbildung 4.12 zu sehen. Sind Sie übrigens bezüglich des eingestellten Datumsformats unsicher, kann ein Blick auf eine solche Ergebnistabelle weiterhelfen. Hier sehen Sie unmittelbar die Form „jjjj-mm-tt“, also vierstelliges Jahr, Bindestrich, Monat, Bindestrich und schließlich der Tag. Doch seien Sie vorsichtig, man kann die Datumsdarstellung auch abweichend von dem internen Format einstellen, und dann hilft nur noch probieren, ein Blick in die Einstellungen der Datenbank oder ein Anruf beim Administrator.

Info

Neben dem unterschiedlichen Format sind hier auch unterschiedliche Platzhalter verwendet worden. MS Access nutzt das Windows-typische *-Zeichen, während MySQL das SQL-standardkonforme %-Zeichen verwendet.

Oracle, Firebird und openBase verwenden dieselben standardkonformen Platzhalter wie MySQL. openBase kennt beide Varianten.

Mehr zu den Platzhaltern finden Sie im Anschluss an die Operatorenliste in Tabelle 4.1.

Abbildung 4.12
Ergebnis der SELECT-Anweisung mit WHERE-Klausel in MySQL

Familienname	Vorname	PLZ	Strasse	Geburtsdatum
Martens	Melanie	29221	Celle	1911-02-17
Peredy	Helmut	29221	Celle	1956-02-23
Sander	NULL	29223	Celle	1953-02-05
Schlachter	Dieter	29227	Celle	1961-02-02

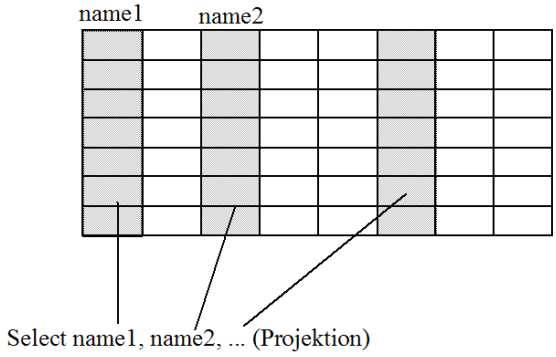
Die WHERE-Klausel beschreibt die Bedingungen, die ein Datensatz erfüllen muss, damit er Teil der Ergebnismenge einer SELECT-Anweisung wird. Sie besteht also aus einer Liste von Bedingungen. Jede Bedingung liefert entweder den Wert WAHR (true) oder FALSCH (false). Diese Bedingungen werden in der Liste mit AND (und) und OR (oder) verknüpft. Ist das Gesamtergebnis für einen Datensatz WAHR, wird er Bestandteil der Ergebnismenge, sonst wird er aussortiert.

Die Logik der Verknüpfungen mit UND, ODER und NICHT und ihr Zusammenspiel werden in Anhang B zur booleschen Logik genauer erläutert.

Projektion

Die SELECT-Klausel mit ihrer Feldnamenliste und die WHERE-Klausel mit ihrer Bedingungsliste ergänzen sich in der Auswahl der anzuzeigenden Zellen einer Tabelle. Das SELECT wählt über die angegebenen Feldnamen die anzuzeigenden Spalten aus einer Tabelle aus, man spricht auch von einer Projektion (siehe Abbildung 4.13).

Abbildung 4.13
Auswahl der Spalten über die SELECT-Klausel



Bei einer Projektion werden also die benötigten Datenfelder einer Tabelle bestimmt. Demgegenüber wird über die Angaben in der WHERE-Klausel festgelegt, welche Zeilen der Tabelle Bestandteil der Datensätze in der Ergebnismenge werden, mengentheoretisch ist dies eine Selektion (siehe Abbildung 4.14).

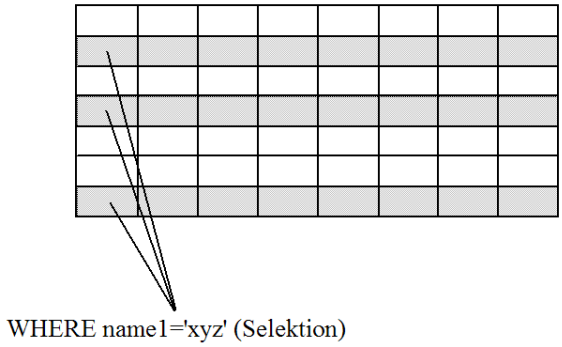


Abbildung 4.14
Auswahl der Datensätze über die WHERE-Klausel

Zweites Beispiel

Schauen wir uns die WHERE-Klausel an einem zweiten Beispiel an. Es soll die Liste aller Kursteilnehmer des Kurses „CE23“ erstellt werden, wobei die Zahlweise, der Rabatt und der bereits gezahlte Betrag von Interesse sind. Teilnehmer, die mit Gutschein bezahlen, sollen nicht berücksichtigt werden. Es soll absteigend nach dem gezahlten Betrag sortiert werden. In SQL ergibt sich dann:

```
SELECT k.KID, k.Zahlweise, k.Rabatt, k.GezahlterBetrag
FROM tbKursbesuche k
WHERE ((k.KID='CE23') AND (k.Zahlweise!='Gutschein'))
ORDER BY k.GezahlterBetrag DESC;
```

Listing 4.20
Teilnehmer des Kurses CE23, die nicht mit Gutschein bezahlen

Info

In MS Access kann das '!=' entsprechend durch ein '<>' ersetzt werden, in anderen Systemen wie DB2 muss es das sogar.

Das Ergebnis der Anweisung ist in Abbildung 4.15 zu sehen. Es sind nur die Teilnehmer in der Ergebnismenge enthalten, für die beide Bedingungen WAHR sind, sie sind Teilnehmer des Kurses „CE23“ UND sie zahlen NICHT mit einem Gutschein.

KID	Zahlweise	Rabatt	GezahlterBetrag
CE23	Bar	0,00 €	350,00 €
CE23	Überweisung	50,00 €	300,00 €
CE23	Bar	100,00 €	250,00 €
CE23	Überweisung	150,00 €	50,00 €
CE23	Überweisung	100,00 €	50,00 €

Abbildung 4.15
Ergebnis der SQL-Anweisung

Zur Ermittlung kritischer Fälle sollen jetzt noch alle Teilnehmer des Kurses „CE23“ ermittelt werden, die bar oder per Überweisung zahlen und noch nicht zwischen 250,- € und 350,- € bezahlt haben oder die den Kurs mit einem Gutschein besuchen und schon mehr als einen Fehltag haben.

Listing 4.21
Kombination von AND und OR

```
SELECT
   k.KID, k.Zahlweise, k.Rabatt,
   k.GezahlterBetrag, k.Fehltage, k.KTID
FROM tbKursbesuche k
WHERE ((k.KID="CE23")
      AND (k.Zahlweise IN ('Bar','Überweisung'))
      AND (k.GezahlterBetrag NOT BETWEEN 250 AND 350))
   OR
      ((k.KID="CE23")
      AND (k.Zahlweise='Gutschein')
      AND (k.Fehltage>1));
```

Die WHERE-Bedingung besteht aus zwei mit OR verbundenen Teilen. Ein Teilnehmer ist in der Ergebnismenge enthalten, wenn er den ersten Teil ODER den zweiten Teil (oder beide, was hier aber logisch nicht möglich ist) erfüllt. Beide Teile bestehen aus mehreren mit AND verbundenen Bedingungen, die jeweils also alle WAHR sein müssen, damit ein Teilnehmer berücksichtigt wird. Das Ergebnis ist in Abbildung 4.16 dargestellt.

Abbildung 4.16
Ergebnis der Abfrage

KID	Zahlweise	Rabatt	GezahlterBetrag	Fehltage	KTID
CE23	Gutschein	0,00 €	350,00 €	2	4
CE23	Überweisung	100,00 €	50,00 €	0	5
CE23	Überweisung	150,00 €	50,00 €	8	37

Hier wird der BETWEEN-Operator verwendet. Mit BETWEEN werden aus allen Datensätzen nur diejenigen Daten ausgewählt, bei denen die Werte des Feldes GezahlterBetrag zwischen „250“ und „350“ liegen. Die beiden Grenzwerte sind eingeschlossen. Zusätzlich ist dem BETWEEN ein NOT vorangestellt. NOT ist ein Operator, der den Wert jedes Ausdrucks negiert, also gerade das umgekehrte Ergebnis liefert. Daher werden die Datensätze mit den KTID „5“ und „37“ berücksichtigt. Sie wären in dem Ausdruck (GezahlterBetrag BETWEEN 250 AND 350) nicht enthalten gewesen und werden dann durch das NOT wieder aufgenommen.

Natürlich wäre hier bei den gegebenen Daten statt des BETWEEN-Ausdrucks auch einfach ein (k.GezahlterBetrag < 250) möglich und wahrscheinlich sinnvoller gewesen, aber so sehen Sie sowohl einen weiteren Operator als auch den Einsatz der Verneinung im Beispiel.

IN

Die SQL-Anweisung enthält noch einen weiteren neuen Operator. Mit dem IN-Operator wird die Aufzählung für die beiden Zahlungsarten berücksichtigt. Bei diesem Kurs wäre sicherlich auch ein (Zahlungsart <> 'Gutschein') beziehungsweise (Zahlungsart != 'Gutschein') möglich gewesen. Kommen später weitere Zahlungsarten hinzu, würde dies aber nicht mehr funktionieren. Mit dem IN-Operator kann dagegen gezielt eine Liste erlaubter Werte definiert werden. Die einzelnen Werte in der Liste des IN-Operators werden jeweils durch ein Komma voneinander getrennt.

MS Access: ';'

Beachten Sie, dass MS Access in Aufzählungen wie dieser abweichend vom SQL-Standard das Semikolon verwendet.

Interessant ist auch die Klammerung der Bedingungen, die mit AND und OR verbunden werden. Jede Bedingung wird zunächst für sich geklammert. Dann werden die ersten drei mit AND verbundenen Bedingungen zusätzlich

noch einmal geklammert, ebenso die drei letzten Bedingungen. Dadurch steht das OR zwischen drei jeweils mit AND verbundenen Bedingungen. Bei der Auswertung des Gesamtausdrucks kann durch die Klammerung die Reihenfolge der Auswertung für den Leser der SQL-Anweisung transparent gemacht werden. Grundsätzlich ist die Reihenfolge der Auswertung durch eine Gewichtung der Operatoren festgelegt und wäre damit eindeutig. Da das AND stärker bindet als das OR, wären diese Klammern hier nicht unbedingt erforderlich gewesen, die Anweisung würde ohne sie genauso funktionieren. Zur besseren Übersicht und Sicherheit ist es aber sinnvoll, in der gegebenen Weise zu klammern.

Bedingungen in MS Access und openBase

Für die Benutzer der MS Access-Oberfläche sowie der openBase-Oberfläche gibt es für die Eingabe der AND- und OR-Verknüpfungen neben der Möglichkeit, diese direkt einzugeben, die Besonderheit, dies durch geschickte Verteilung der Zeilen für die Auswertungskriterien zu erreichen.

Feld:	KID	Zahlweise	Rabatt	GezahlterBetrag	Fehltage	KTID
Tabelle:	tbKursbesuche	tbKursbesuche	tbKursbesuche	tbKursbesuche	tbKursbesuche	tbKursbesuche
Sortierung:						
Anzeigen:	☑	☑	☑	☑	☑	☑
Kriterien:	="CE23"	In ('Bar';'Gutschein')		Nicht Zwischen 250 Und 350		
oder:	="CE23"	'Gutschein'				

Abbildung 4.17
MS Access-Eingabe für obige Werte

Die Eingabe der Bedingungen für die WHERE-Klausel erfolgt in MS Access (und ähnlich in openBase) über die Zeile KRITERIEN und die folgenden Zeilen. In die Spalte des jeweiligen Feldes wird dabei die Bedingung unmittelbar eingetragen (siehe Abbildung 4.17). Beachten Sie die Spracheinstellung.

Aus den Einträgen in diesen Zeilen generiert MS Access dann die WHERE-Klausel. Dabei erzeugt MS Access aus jeder Zeile eine Reihe von Bedingungen, die mit einem AND verbunden werden. Eine Zeile beschreibt also eine Reihe von Bedingungen, die alle erfüllt sein müssen. Anschließend werden die einzelnen Zeilen mit einem OR verbunden. Somit ergibt sich genau die Struktur, die wir oben im Zusammenhang mit der Klammerung angesprochen haben.

Bedingungsliste

Die WHERE-Klausel besteht also aus dem Schlüsselwort WHERE und einer darauffolgenden Bedingungsliste. Der Aufbau der Bedingungsliste selbst ist zunächst relativ einfach. Jede Bedingungsliste folgt der Struktur

(*bedingung***)** { *AND* | *OR (bedingung)* }

mit mindestens einer und (theoretisch) beliebig vielen Bedingungen, die durch AND- und OR-Operatoren miteinander verbunden werden.

Die einzelnen Bedingungen müssen – entgegen obiger Darstellung – nicht zwingend in Klammern gesetzt werden, dies erhöht aber die Lesbarkeit in vielen Fällen und verhindert Mehrdeutigkeiten. Daher sind die obigen Klammern als dringende Empfehlung für die zu verwendende Syntax zu verstehen. Generell gilt, dass AND stärker bindet als OR.

Bedingung

Jede einzelne Bedingung hat das Format:

(*feldname Operator ausdruck)*

Die zugelassenen Operatoren, die auch als *Prädikate* bezeichnet werden, sind der Tabelle 4.1 zu entnehmen.

Tabelle 4.1 Operatoren (Prädikate) der WHERE-Klausel

Operator	Bedeutung	Beispiel
`=`	Prüft auf Gleichheit (bei MySQL auch `<=>`)	`Ort = 'Celle'`
`!=, <>`	Prüft auf Ungleichheit. `<>` wird nicht von allen RDBMS unterstützt, DB2 nur `<>`	`Ort != 'Celle'` `Ort <>'Celle'`
`<`	Prüft auf „kleiner als“	`Stundensatz < 15` `PLZ < '30000'`
`<=`	Prüft auf „kleiner“ oder Gleichheit	`Stundensatz <= 15`
`>`	Prüft auf „größer als“	`Stundensatz > 15` `PLZ >'28999'`
`>=`	Prüft auf „größer“ oder Gleichheit	`Stundensatz >= 15` `PLZ >= '29000'`
`IS [NOT] NULL`	Prüft auf das Fehlen eines Wertes (`NULL`-Wert). Dies ist nicht gleichwertig mit `=''` oder `=0` oder ähnlichen Abfragen, denn in diesen Fällen wird mit Werten verglichen. Die Antwort auf `IS NULL` ist für jeden von `NULL` verschiedenen Wert falsch, also auch beispielsweise für 0 oder ein leeres Feld.	`Vorname IS NULL` `Geburtsdatum IS NOT NULL`

`[NOT] LIKE`	Fragt nach Mustern mithilfe von Platzhaltern (`'%'`, `'_'` bzw. `'*'` und `'?'`) (siehe unten) ab. Damit lassen sich beispielsweise Felder finden, die einen bestimmten Text enthalten, mit diesem beginnen oder enden.	`PLZ LIKE '29%'`
		`Name = '_eiss'`
		`Vorname = 'Ka___'`

`NOT [BETWEEN]`	Prüft, ob der Feldwert in einen bestimmten Wertebereich fällt. Die Grenzen gehören zum gültigen Bereich.	`PLZ BETWEEN 29000 AND 29999`
		entspricht
		`PLZ >= 29000 AND`
		`PLZ <= 29999`

Tabelle 4.1 (Forts.) Operatoren (Prädikate) der WHERE-Klausel

Operator	Bedeutung	Beispiel
`[NOT] IN`	Prüft, ob der Feldwert in einer gegebenen Menge von Werten auftritt.	`Ort IN ('Braunschweig', 'Hannover')`
`[NOT] EXISTS`, `ALL`, `ANY`	Überprüft jeweils für einen gesamten Datensatz, also nicht für ein einzelnes Feld, ob eine zumeist mit einer Unterabfrage formulierte Bedingung zutrifft.	siehe Kapitel 9, Unterabfragen
Reguläre Ausdrücke	Reguläre Ausdrücke entsprechen weitgehend den von LINUX/UNIX bekannten Ausdrücken. Sie sind unter MySQL, Oracle, SQL Server und einigen anderen Datenbanken verfügbar.	`[23][0-9]+` Postleitzahlen, die mit 2 oder 3 beginnen.

Achtung

An verschiedenen Stellen können Platzhalter, sogenannte Wildcards, eingesetzt werden, die einzelne unbekannte Zeichen oder ganze Zeichenfolgen repräsentieren.

In Windows haben sich die Platzhalter `*` für eine beliebig lange (auch leere) Zeichenkette und `?` für ein einzelnes Zeichen durchgesetzt.

SQL hat derartige Platzhalter schon wesentlich früher eingeführt, Windows hat aber dann später die SQL-Platzhalter leider nicht übernommen.

In SQL ist das Prozentzeichen (`%`) für eine beliebig lange, auch leere Zeichenkette üblich. Ein einzelnes Zeichen, das auch fehlen kann, wird durch einen Unterstrich _ repräsentiert.

Also für Windows-Nutzer: `*` entspricht `%` und `?` entspricht _.

Beispiel: Statt `'29*'` muss `'29%'` angegeben werden, statt `'2922?'` ist `'2922_'` zu schreiben.

Beispiele für Ausdrücke mit Platzhaltern (Wildcards):

- `M%`: alle Namen, die mit M anfangen
- `%er`: alle Namen, die mit er enden
- `M%er`: alle Namen, die mit M anfangen und mit er enden
- `M_ier`: alle Namen, bei denen der Unterstrich ersetzt werden kann, beispielsweise Meier oder Maier

Zusammenfassung

Sie haben jetzt gesehen, wie die in der Ergebnismenge einer SELECT-Anweisung enthaltenen Datensätze ausgewählt werden können. Man nennt dies auch filtern. Wir haben unsere Kenntnisse der SELECT-Syntax damit wieder erweitert:

Syntax mit WHERE-Klausel

SELECT [DISTINCT|ALL] *feldname [AS alias]* {,*feldname [AS alias]*}

FROM *tabellenname [AS alias]*

[WHERE (*bedingung)* { *AND* | *OR (bedingung)* } *]*

[ORDER BY *feldname [ASC|DESC]* { ,*feldname [ASC|DESC]*}*];*

Die WHERE-Klausel kann in einer SELECT-Anweisung fehlen. Wenn sie vorhanden ist, muss sie mindestens eine Bedingung enthalten. Eine Bedingung ist ein Ausdruck, der WAHR oder FALSCH ist und sinnvollerweise zumeist einen Vergleich eines Feldwertes mit einem anderen Wert beinhaltet. Bedingungen sollten geklammert sein und werden mit AND oder OR miteinander verbunden.

Übungen

Übungen zur SELECT-Anweisung mit WHERE-Klausel

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Erstellen Sie eine Liste aller Personen mit Familienname, Vorname, PLZ, Ort und Geburtsdatum, die in Braunschweig wohnen. (Ü4.4.1)
2. Ermitteln Sie alle Personen mit Familienname, Vorname, PLZ, Ort und Geburtsdatum, die aus einem Ort kommen, der mit einem „B“ beginnt. Sortieren Sie die Orte alphabetisch und verwenden Sie dabei die Positionsnummer der Sortierspalte. (Ü4.4.2)
3. Ermitteln Sie alle Personen mit Familienname, Vorname, PLZ, Ort und Strasse, die aus einem Ort kommen, dessen Postleitzahl kleiner als 30000 ist, und die in einer Straße wohnen, die „allee“ enthält. Sortieren Sie die Postleitzahlen aufsteigend. Geben Sie als Alias für das PLZ-Feld „kleiner 30000“ an. Qualifizieren Sie alle Angaben. (Ü4.4.3)
4. Ermitteln Sie in der Tabelle tbKursthema alle Kursthemen, in deren Kursbeschreibung die Begriffe „Access“, „Excel“ oder „Datenbank“ vorkommen und die 40 oder 80 Stunden dauern. Zeigen Sie nur die Themen an. (Ü4.4.4)
5. Ermitteln Sie aus der Tabelle tbKursbesuche alle Teilnehmer des Kurses „CE23“ mit KID, Zahlweise, Rabatt und dem gezahlten Betrag, die entweder mit Gutschein bezahlen oder nicht mit Gutschein bezahlen und mindestens 250,- € bezahlt haben. Sortieren Sie die Liste absteigend nach dem gezahlten Betrag. (Ü4.4.5)

4.5 Tabellen miteinander verbinden (JOIN)

4.5.1 Der Klassiker (INNER JOIN)

Wir konnten bisher recht komfortabel Daten aus den Kursbesuchen filtern, um beispielsweise zu ermitteln, wer seine Gebühren noch nicht in nennenswertem Umfang bezahlt hat. Dabei haben wir aber für jeden Kursteilnehmer stets nur die KTID, die Kursteilnehmer-Identifikationsnummer, angegeben. Schön wäre es, wenn Si auch seinen Namen und die Anschrift ermitteln könnten. Da es sich bei dem Fremdschlüssel KTID zugleich um den Primärschlüssel PID der Personentabelle tbPerson handelt, können Sie dort natürlich leicht die weiteren Daten zu den Personen ermitteln, indem Sie einfach mit der ermittelten Nummer suchen. Sie müssen dafür aber beide Tabellen miteinander verbinden.

Wir wollen daher jetzt Daten aus mehreren Tabellen verbinden, um so zu umfassenderen Informationen zu gelangen. Wir betrachten nicht mehr isolierte Tabellen, sondern Zusammenhänge zwischen verschiedenen Daten und können so übergreifende Daten auswerten.

Beispiel

Lassen Sie uns das oben beschriebene Problem, weitere Informationen über Personen zu ergänzen, ausarbeiten. Wir wollen dabei den Namen und die Adresse aller Kursteilnehmer bestimmen, die noch keine 250 € bezahlt haben. Dazu sind grundsätzlich zwei Arbeitsschritte notwendig:

1. Zunächst werden in der Tabelle tbKursbesuche die Datensätze mit geringeren Zahlungen mithilfe einer SELECT-Anweisung ermittelt.

Fremdschlüssel = Primärschlüssel

2. Dann werden dem Ergebnis der Anweisung die Werte aus dem Feld KTID (= PID der Personentabelle) als Fremdschlüssel entnommen. Die so ermittelten Werte werden dann in einer zweiten SQL-Anweisung in der PID als Primärschlüssel in der Tabelle tbPerson gesucht. Es gehören immer die Datensätze aus tbPerson und aus tbKursbesuche zusammen, deren Werte in dem Feld PID beziehungsweise KTID übereinstimmen. Haben Sie die zusammengehörigen Datensätze ermittelt, ist es nicht mehr schwer, die weiteren Daten aus den übrigen Feldern der zusammengehörenden Datensätze zu bestimmen. Was Sie tun müssen, ist also den Fremdschlüssel KTID der Tabelle tbKursbesuche mit dem Primärschlüssel PID der Tabelle tbPerson gleichzusetzen. Das nennt man in SQL einen JOIN, der sich auch direkt in einer SQL-Anweisung ausführen lässt.

Beispiel

Sollen beispielsweise die Adressen aller Personen ermittelt werden, die am Kurs „CE23“ teilnehmen und noch nicht mindestens 250 € Kursgebühr bezahlt haben, so können Sie beide Schritte in einer SELECT-Anweisung zusammenfassen.

Listing 4.22
Beispiel für einen JOIN

```
SELECT
   k.KID, k.GezahlterBetrag, k.KTID, p.PID,
   p.Familienname, p.Vorname, p.PLZ, p.Ort, p.Strasse
FROM tbPerson p INNER JOIN tbKursbesuche k ON (p.PID = k.KTID)
WHERE ((k.KID='CE23') AND (k.GezahlterBetrag<250))
ORDER BY p.Familienname ASC, p.Vorname ASC;
```

Das Ergebnis ist dann in Abbildung 4.18 zu sehen. Sie sehen, dass in jedem „Datensatz“ des Ergebnisses Werte aus zwei tatsächlichen Datensätzen der beiden Tabellen kombiniert worden sind. Diese Datensätze haben identische Werte in den Feldern `KTID` beziehungsweise `PID`, hier also jeweils „5“ oder „37“.

Abbildung 4.18
Ergebnis der SELECT-Anweisung

KID	GezahlterBetrag	KTID	PID	Familienname	Vorname	PLZ	Ort	Strasse
CE23	50,00 €	5	5	Klötzer	Karl	29221	Celle	Bahnhofstraße 2
CE23	50,00 €	37	37	Magerkurth	Melissa	29336	Nienhagen	Am Tümpel 3

Der `JOIN` kann auch erfolgen, ohne dass das Fremdschlüsselfeld und/oder das Primärschlüsselfeld selbst Bestandteil der Ergebnismenge ist. Obiges Beispiel hätte man also auch vereinfachen können, wie in der folgenden `SELECT`-Anweisung dargestellt:

Listing 4.23
JOIN ohne Ausgabe der Schlüsselfelder

```
SELECT
   k.KID, k.GezahlterBetrag, p.Familienname,
   p.Vorname, p.PLZ, p.Ort, p.Strasse
FROM tbPerson p INNER JOIN tbKursbesuche k ON p.PID = k.KTID
WHERE ((k.KID='CE23') AND (k.GezahlterBetrag<250))
ORDER BY p.Familienname ASC, p.Vorname ASC;
```

Das Ergebnis ist dann in Abbildung 4.19 zu sehen.

Abbildung 4.19
Ergebnis der SELECT-Anweisung

KID	GezahlterBetrag	Familienname	Vorname	PLZ	Ort	Strasse
CE23	50,00 €	Klötzer	Karl	29221	Celle	Bahnhofstraße 2
CE23	50,00 €	Magerkurth	Melissa	29336	Nienhagen	Am Tümpel 3

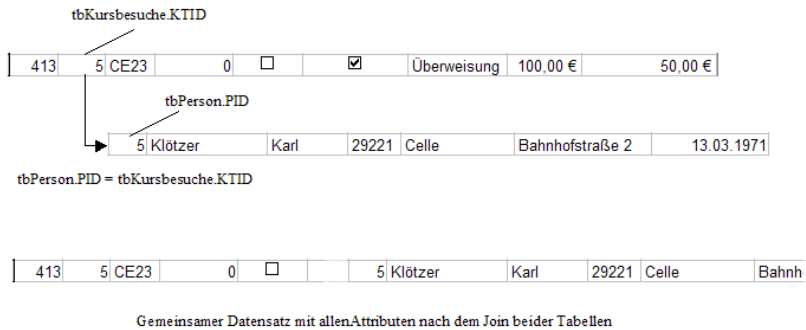
Hier sind zum ersten Mal zwei Tabellen in einer `SELECT`-Anweisung angesprochen: `tbPerson` alias `p` und `tbKursbesuche` alias `k`. Entsprechend kann die Ergebnismenge aus einer Auswahl aus den Feldern dieser beiden Tabellen bestehen. `KID` und `GezahlterBetrag` sind der Tabelle `tbKursbesuche` entnommen, der Rest der Tabelle `tbPerson`. Die Qualifizierung der Feldnamen macht die SQL-Anweisung wieder einmal deutlich lesbarer.

Neu ist in beiden SQL-Anweisungen der Ausdruck

```
FROM tbPerson p INNER JOIN tbKursbesuche k ON p.PID = k.KTID
```

Übersetzt bedeutet diese Zeile: „Verbinde die Datensätze der beiden Tabellen `tbPerson` und `tbKursbesuche` so, dass ein neuer Datensatz entsteht, bei dem der Wert des Feldes `PID` in `tbPerson` gleich dem Wert des Feldes `KTID` der Tabelle `tbKursbesuche` ist (siehe Abbildung 4.20).“

Abbildung 4.20
Zusammenführung zweier Datensätze über Fremdschlüssel und Primärschlüssel


tbKursbesuche.KTID
413 5 CE23 0 Überweisung 100,00 € 50,00 €
tbPerson.PID
5 Klötzer Karl 29221 Celle Bahnhofstraße 2 13.03.1971
tbPerson.PID = tbKursbesuche.KTID
413 5 CE23 0 5 Klötzer Karl 29221 Celle Bahnh
Gemeinsamer Datensatz mit allenAttributen nach dem Join beider Tabellen

Ein JOIN erzeugt also neue virtuelle Datensätze. Es entsteht eine neue virtuelle Tabelle, die alle Datenfelder beider Tabellen enthält. Dabei werden jeweils die beiden Datensätze aus den Ausgangstabellen zu einem neuen Datensatz kombiniert, deren Fremdschlüssel beziehungsweise Primärschlüssel denselben Wert haben. Stellen Sie sich einfach eine neue Tabelle vor, mit der Sie weiterarbeiten können, als ob sie wirklich vorhanden wäre. Dieses Denkmodell wird in Abbildung 4.21 visualisiert. Es gilt:

- Wenn der Fremdschlüsselwert einer Tabelle mit dem Primärschlüsselwert der anderen Tabelle übereinstimmt, entsteht ein neuer gemeinsamer Datensatz.
- Wenn zu einem Fremdschlüsselwert kein Primärschlüsselwert in der anderen Tabelle existiert, bleibt der Datensatz unberücksichtigt.
- Wenn umgekehrt zu einem Primärschlüsselwert kein Fremdschlüsselwert existiert, bleibt auch dieser Datensatz unberücksichtigt.
- Es entstehen also genauso viele Datensätze, wie es passende Kombinationen beider Tabellen gibt.

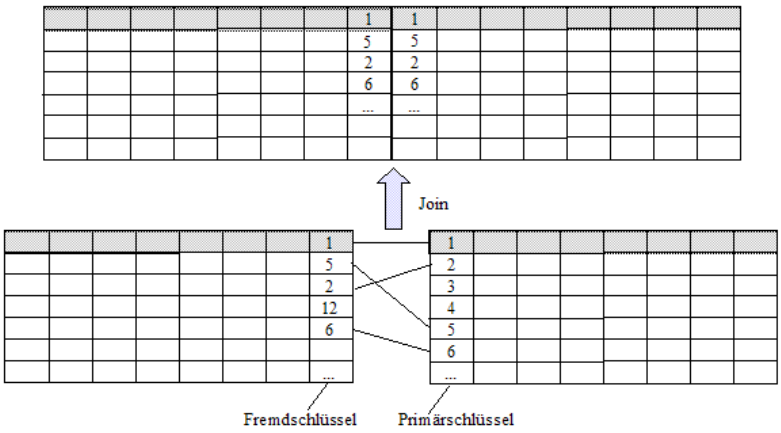


Abbildung 4.21
Zusammenführen zweier Tabellen zu einer virtuellen Tabelle mit einem JOIN

Dieses Denkmodell ist bei der Definition eines JOIN hilfreich. Stellen Sie sich jeden JOIN als materialisierte – virtuelle – Tabelle vor. Auch wenn der SQL-Interpreter in Wirklichkeit wegen verschiedener Optimierungen selten (wahrscheinlich nie) diese Tabelle komplett erstellt, sind dies nur technische Optimierungen. Aus logischer Sicht – aus unserer SQL-Sicht – verhält sich der JOIN wie eine solche temporär erzeugte Tabelle, auf der dann weiter gearbeitet wird. Wenn Sie mit Ihrer SELECT-Anweisung das gewünschte Ergebnis erreicht haben, können Sie immer noch über die tatsächliche Arbeitsweise des SQL-Interpreters nachdenken und die SELECT-Anweisung optimieren. Sie kennen die alte Programmierregel: „Make it right before you make it faster!“ Also machen wir es erst einmal richtig, bevor wir über Performance nachdenken.

INNER JOIN-Syntax

Nachdem wir jetzt einen ersten Blick auf einen JOIN geworfen haben, ist es an der Zeit, die Syntax näher zu analysieren. Beachten Sie aber, dass der JOIN in sehr unterschiedlichen Formen auftreten kann. Wir wollen uns zunächst auf die gängigste Form – den sogenannten INNER JOIN – in der Fassung von SQL92 konzentrieren.

FROM *tabellenname1*

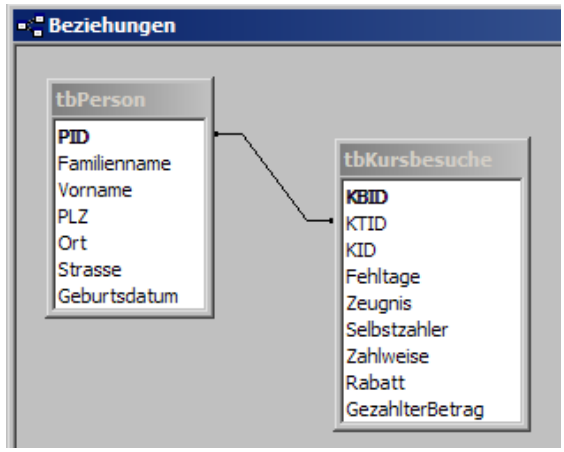
{ [INNER] JOIN *tabellenname2*
ON tabellenname1.feld1 = tabellenname2.feld2}

Die Angabe INNER ist theoretisch nicht notwendig, darauf sollten Sie sich in der Praxis der gängigen Datenbankversionen allerdings nicht verlassen. Sie sollten daher wenn möglich stets INNER JOIN schreiben. Dies hilft außerdem bei der Unterscheidung von dem später zu besprechenden OUTER JOIN.

Beziehung

Der JOIN beruht immer auf der Verbindung zweier Tabellen über (zumindest) ein Paar von Feldern. Dies lässt sich auch auf Tabellenebene darstellen. MS Access verwendet dazu das sogenannte Beziehungsfenster, in dem (mögliche) Verbindungen dargestellt werden können. Damit lässt sich die Struktur der Datenbank visualisieren, die in allen Datenbanken existiert.

Abbildung 4.22
Verbindung zweier Tabellen durch INNER JOIN über PID = KTID



Weiteres Beispiel

Mithilfe eines JOIN lässt sich beispielsweise auch eine Kursliste erstellen, die neben der Kurskennung und dem Anfangs- und Endedatum des Kurses das Kursthema angibt. Dazu werden die beiden Tabellen tbKurs und tbKursthema miteinander verbunden:

Listing 4.24
SQL-Anweisung für eine Kursliste

```
SELECT k.Kurskennung, k.Kursbeginn, k.Kursende, kt.Kursthema
FROM tbKurs k INNER JOIN tbKursthema kt ON k.KTHID = kt.KTHID
ORDER BY k.Kursbeginn ASC;
```

Das Ergebnis des JOIN ist in Abbildung 4.23, diesmal als Oracle-Ergebnis, dargestellt.

KURSKENNUNG	KURSBEGINN	KURSENDE	KURSTHEMA
Celle17-Word	23.04.07	27.04.07	Word I
Celle20-Word	09.07.07	13.07.07	Word I
Hannover89-Access	09.07.07	13.07.07	Access I
Hannover90-Windows	16.07.07	18.07.07	Einstieg mit Windows
Hannover91-Windows	23.07.07	25.07.07	Windows für Fortgeschrittene
Celle23-Access	06.08.07	10.08.07	Access I
Celle24-Access	13.08.07	17.08.07	Access II

7 rows returned in 0,11 seconds CSV Export

Abbildung 4.23
Kurse mit Kursthema

Die „alte“ JOIN-Syntax

JOIN mit SQL89

Leser, die sich früher schon einmal mit SQL beschäftigt haben, werden sich vielleicht über die bisher beschriebene `JOIN`-Syntax gewundert haben. Tatsächlich war bis zur Verabschiedung von SQL92 ausschließlich eine andere Syntax erlaubt und gebräuchlich. Dabei wurde der `JOIN` ebenfalls über die Gleichheit der Werte zweier Felder durchgeführt. Dieser `JOIN` war allerdings nicht Bestandteil der `FROM`-Klausel, sondern der `WHERE`-Klausel, in der die Felder einfach gleichgesetzt wurden. Die in Listing 4.22 vorgestellte `SELECT`-Anweisung würde man dann wie folgt formulieren:

Listing 4.25
JOIN über die WHERE-Klausel

```
SELECT
   k.KID, k.GezahlterBetrag, p.Familienname,
   p.Vorname, p.PLZ, p.Ort, p.Strasse
FROM tbPerson p, tbKursbesuche k
WHERE ((p.PID = k.KTID)
   AND (k.KID='CE23')
   AND (k.GezahlterBetrag<250))
ORDER BY p.Familienname, p.Vorname;
```

Sie sehen, dass in der `FROM`-Klausel nur die zu verwendenden Tabellen aufgezählt, aber keinerlei Angaben zum `JOIN` gemacht werden. Die Gleichsetzung der beteiligten Felder, also der `PID` aus der Tabelle `tbPerson` und der `KTID` aus der Tabelle `tbKursbesuche` erfolgt ausschließlich in der `WHERE`-Klausel. Diese Syntax ist bis heute in allen gängigen relationalen Datenbankmanagementsystemen gültig und kann gleichwertig mit der neueren `JOIN`-Syntax in der `FROM`-Klausel verwendet werden.

Egal, welche Syntax verwendet wird, beruht der `JOIN` immer auf der Verbindung zweier Tabellen über (zumindest) ein gemeinsames Paar von Feldern.

Tipp

Verwenden Sie wenn möglich die neue SQL92-Syntax. Sie ist eindeutig und erlaubt dem SQL-Interpreter wegen der zusätzlichen Information eine bessere interne Optimierung.

Self-JOIN

Ein `JOIN` kann auch auf einer Tabelle gemacht werden. Dabei werden gedanklich zwei Kopien der Tabelle verwendet, die über einen `JOIN` miteinander verbunden werden. Die Tabelle enthält gleichzeitig Primärschlüssel

und Fremdschlüssel und ihre beiden gedachten Kopien werden über diese beiden Felder miteinander verbunden. Man spricht von einem *rekursiven JOIN* oder *Self-JOIN*. So ist in der Tabelle `tbKursthema` zu einigen Kursthemen auch ein weiteres Kursthema als Voraussetzung für den Besuch des Kurses angegeben. Mithilfe eines `JOIN` kann jetzt eine Liste erstellt werden, die zu jedem Kurs dessen Voraussetzung mit angibt.

Listing 4.26
JOIN auf der Tabelle tbKursthema

```
SELECT
   kt.KTHID AS "ID",
   kt.Kursthema AS "Thema",
   kv.KTHID AS "Voraussetzung ID",
   kv.Kursthema AS "Voraussetzung Thema"
FROM tbKursthema kt INNER JOIN tbKursthema kv
   ON (kv.KTHID = kt.Voraussetzung)
ORDER BY kt.KTHID ASC;
```

Es wird zweimal die Tabelle `tbKursthema` angesprochen. Um beide gedachten „Tabellen" unterscheiden zu können, ist die Verwendung von Alias zwingend. Hier wird das Thema selbst mit dem Alias `kt`, dessen Voraussetzung mit dem Alias `kv` bezeichnet. Entsprechend können dieselben Datenfelder je Alias getrennt verwendet werden. Sinnvoll ist es dabei auch, für die Datenfeldnamen Alias zu verwenden, um sie in der Ausgabe besser unterscheiden zu können.

Beachten Sie, dass die Liste nur Kurse enthält, die tatsächlich einen anderen Kurs als Voraussetzung haben. Sollen alle Kurse angegeben werden, also auch die Kurse ohne Voraussetzung, muss ein `OUTER JOIN` verwendet werden, worauf wir noch zurückkommen werden.

Abbildung 4.24
Kurse mit ihren Voraussetzungen als Self-JOIN

ID	Thema	Voraussetzung ID	Voraussetzung Thema
2	Windows für Fortgeschrittene	1	Einstieg mit Windows
3	Ordnung im Dateisystem	2	Windows für Fortgeschrittene
4	Word I	2	Windows für Fortgeschrittene
5	Access I	2	Windows für Fortgeschrittene
6	Excel I	2	Windows für Fortgeschrittene
7	Access II	5	Access I
8	Datenbankentwurf	7	Access II
9	VBA-Datenbankprogrammierung	7	Access II
10	Word II	4	Word I
11	Excel II	6	Excel I

Übungen

Übungen zur SELECT-Anweisung mit INNER JOIN über zwei Tabellen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Ermitteln Sie für alle Kurse die Kennung, das Kursthema sowie die geplante und die tatsächliche Kursdauer. Verwenden Sie sinnvolle Alias für die Datenfelder und sortieren Sie das Ergebnis alphabetisch nach der Kurskennung. (Ü4.5.1.1)
2. Erstellen Sie eine alphabetische Liste aller Dozenten mit `Familienname`, `Vorname`, `PLZ`, `Ort` und `Stundensatz`. (Ü4.5.1.2)

3. Ermitteln Sie für alle Dozenten eine Liste mit `DID`, `Stundensatz` sowie `Kurskennung`, `Kursbeginn` und `Kursende` der Kurse, die sie leiten. Sortieren Sie die Liste aufsteigend nach dem Kursbeginn. (Ü4.5.1.3)
4. Ermitteln Sie in der Tabelle `tbKursthema` alle eindeutigen Kursthemen, in deren Kursbeschreibung die Begriffe „Access“, „Excel“ oder „Datenbank“ vorkommen und die tatsächlich 40 Stunden dauern. (Ü4.5.1.4)

4.5.2 JOIN über mehrere Tabellen

Ein `JOIN` betrifft zunächst stets zwei Tabellen (oder zwei Kopien einer Tabelle), die miteinander in Verbindung gebracht werden. Natürlich können auch mehr als zwei Tabellen miteinander kombiniert werden, wie Abbildung 4.26 zeigt.

Wir verbinden jetzt drei und mehr Tabellen über mehrere `JOIN`-Verknüpfungen. Damit können wir beliebig viele Tabellen einer Datenbank in einer `SELECT`-Anweisung verwenden und haben vollen Zugriff auf die in einer Datenbank enthaltenen Informationen.

Beispiel

Es soll beispielsweise zu einem Kurs eine Liste der Teilnehmer mit Adressen und Kurskennung erstellt werden. Dafür müssen die drei Tabellen `tbPerson`, `tbKursbesuche` und `tbKurs` miteinander in Beziehung gebracht werden. Die Kurskennung kann letztlich nur aus der Tabelle `tbKurs` ermittelt werden. Die Adressdaten stehen in `tbPerson`. Die Tabelle `tbKursbesuche` verbindet beide Tabellen miteinander. Schränken wir das Ergebnis noch darauf ein, dass eine Liste mit `Kurskennung`, `Familienname` und `Vorname` des Teilnehmers von Kurs „CE23“ erstellt werden soll, kann das als `SELECT`-Anweisung wie folgt umgesetzt werden:

Listing 4.27
JOIN über drei Tabellen

```
SELECT k.Kurskennung, p.Familienname, p.Vorname
FROM tbKurs k INNER JOIN
   (tbPerson p INNER JOIN tbKursbesuche kb ON (p.PID = kb.KTID))
   ON k.KID = kb.KID
WHERE (k.KID = 'CE23')
ORDER BY p.Familienname, p.Vorname;
```

Das Ergebnis der SQL-Anweisung ist in Abbildung 4.25 dargestellt.

Abbildung 4.25
Ergebnis der SELECT-Anweisung

Kurskennung	Familienname	Vorname
Celle23-Access	Karmann	Thomas
Celle23-Access	Klötzer	Karl
Celle23-Access	Magerkurth	Melissa
Celle23-Access	Martens	Melanie
Celle23-Access	Müller	Claudia
Celle23-Access	Ruppert	Nicola
Celle23-Access	Schulze	Tanja
Celle23-Access	Winter	Petra

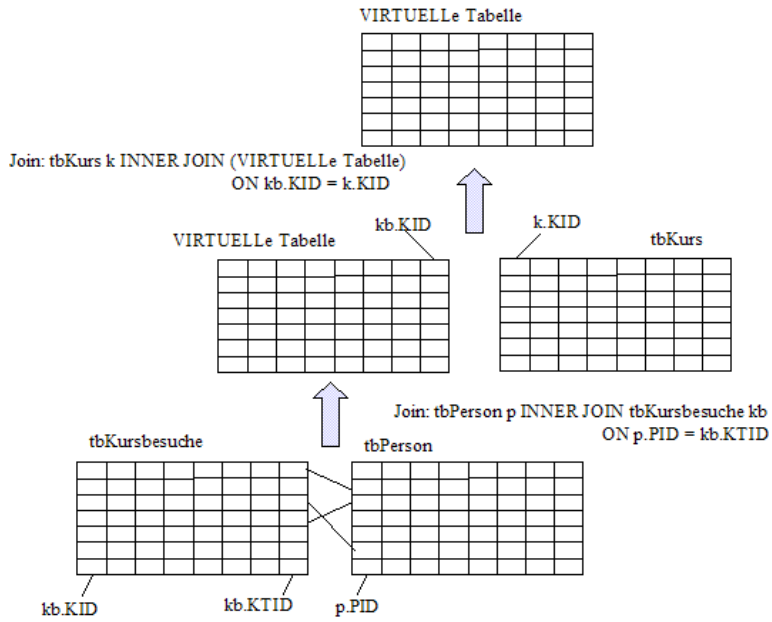
Info

Um noch einmal auf das Thema Qualifizierung zurückzukommen. Spätestens hier ist die Qualifizierung zumindest des Feldes `KID` zwingend. Eine Spalte `KID` tritt in beiden Tabellen auf. Erst durch die Angabe `k.KID` beziehungsweise `kb.KID` wird für den SQL-Interpreter deutlich, dass er das Feld der Tabelle `tbKurs` respektive `tbKursbesuche` entnehmen soll. Also im Zweifelsfall besser qualifizieren.

Die Darstellung in Abbildung 4.26 zeigt die Logik der angegebenen `SELECT`-Anweisung. In einem ersten Schritt werden die Tabellen `tbPerson` und `tbKursbesuche` miteinander zu einer virtuellen Tabelle verbunden. Dieser `JOIN` geht über die Gleichheit der Werte in den Spalten `tbPerson.PID = tbKursbesuche.KTID`.

Dabei gehen alle Spalten beider Tabellen in die resultierende virtuelle Tabelle ein. Dazu gehört auch die Spalte `kb.KID`, die dann im zweiten `JOIN` mit der Tabelle `tbKurs` genutzt wird, um den `JOIN` über `tbKursbesuche.KID = tbKurs.KID` durchführen zu können.

Abbildung 4.26
Zwei aufeinander aufbauende JOIN



Bleibt man bei der Vorstellung der virtuellen Tabellen, kann man den `JOIN` über drei Tabellen wiederum symbolisch darstellen, wie in Abbildung 4.27 zu sehen. Dabei können Sie auch gut die Felder aus den verschiedenen Tabellen erkennen. Die Darstellung zeigt zunächst die mögliche Kombination an. In Abbildung 4.26 sehen Sie aber bereits, dass der `JOIN` nacheinander, nicht gleichzeitig erfolgt. Tatsächlich ist die Reihenfolge, in der die Tabellen miteinander verbunden werden, für das Ergebnis eines `INNER JOIN` nicht relevant, für dessen Performance kann sie aber entscheidend sein.

Daher ist es wichtig, wenn ein JOIN einmal funktioniert, über mögliche Optimierungen nachzudenken.

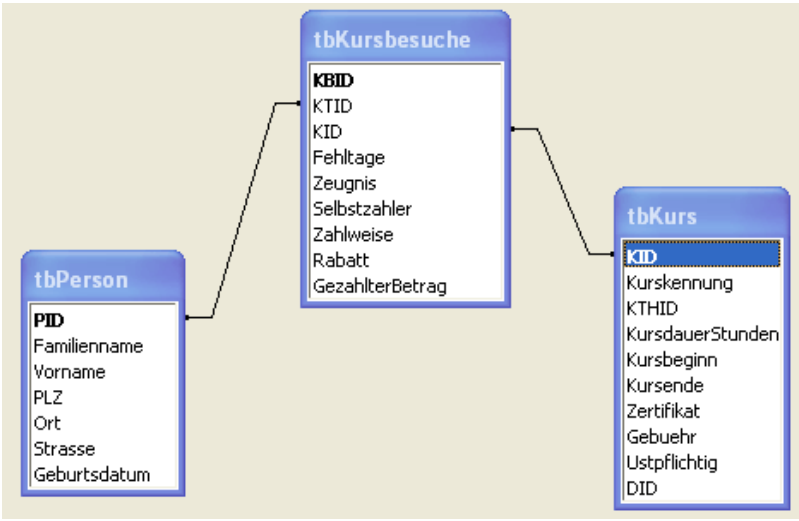


Abbildung 4.27
Verbindung dreier Tabellen

Die Reihenfolge, in der die JOIN-Verknüpfungen gebildet werden, wird dabei durch eventuelle Klammern in der FROM-Klausel bestimmt. Im obigen Beispiel wird zunächst der geklammerte Ausdruck

```
(tbPerson p INNER JOIN tbKursbesuche kb ON p.PID = kb.KTID)
```

als erster Ausdruck ausgewertet, das Ergebnis ist eine virtuelle Tabelle. In einem zweiten Schritt wird dann diese virtuelle Tabelle mit tbKurs verbunden:

```
FROM tbKurs k INNER JOIN (VIRTUELL) ON k.KID = VIRTUELL.KID
```

Solange INNER JOIN-Verknüpfungen verwendet werden, ist die Reihenfolge für das Ergebnis des JOIN letztlich egal. Es könnte also dasselbe Ergebnis auch mit folgender SELECT-Anweisung erreicht werden:

```
SELECT k.Kurskennung, p.Familienname, p.Vorname
FROM (tbKurs k INNER JOIN tbKursbesuche kb ON (k.KID = kb.KID))
   INNER JOIN tbPerson p ON (kb.KTID = p.PID)
WHERE (k.KID='CE23')
ORDER BY p.Familienname, p.Vorname;
```

Listing 4.28
Veränderte JOIN-Reihenfolge

Beachten Sie die veränderte Stellung der Klausel ON (k.KID = kb.KID)in dieser Anweisung. Wie gesagt, das Ergebnis ist das Gleiche und in Abbildung 4.25 dargestellt.

Eine weitere Möglichkeit dies mit einem SELECT zu formulieren, besteht darin, scheinbar zweimal direkt von der mittleren Tabelle tbKursbesuche auszugehen:

Listing 4.29
Dritte Variante des JOIN

```
SELECT k.Kurskennung, p.Familienname, p.Vorname
FROM tbKursbesuche kb
   INNER JOIN tbKurs k ON (k.KID = kb.KID)
   INNER JOIN tbPerson p ON (kb.KTID = p.PID)
WHERE (k.KID='CE23')
ORDER BY p.Familienname, p.Vorname;
```

Performance

Bei der Verwendung eines JOIN mit großen Tabellen muss im Einzelfall die Performance der SELECT-Anweisung analysiert werden. Eine grobe Richtlinie ist dabei, dass zunächst möglichst kleine Tabellen in den JOIN einbezogen werden sollen und die großen Tabellen möglichst spät berücksichtigt werden. Dahinter verbirgt sich die Erkenntnis, dass die virtuellen Zwischentabellen verwaltet werden müssen. Je kleiner diese Tabellen sind, desto performanter ist zumeist der JOIN. Zu beachten ist dabei aber auch, inwieweit über eine WHERE-Klausel eine der am JOIN beteiligten Tabellen bereits „verkleinert" wird. Tatsächlich kann aber auch aufgrund von Indizes und anderen Optimierungen ein anderes Verhalten auftreten, sodass im Zweifelsfall einfach getestet werden sollte, welche Variante günstiger ist.

Klammersetzung

Was passiert aber, wenn beim JOIN mit mehreren Tabellen gar keine Klammern gesetzt werden? Der SQL-Standard sieht vor, dass ein JOIN immer von links nach rechts abgearbeitet wird, dass also streng genommen die Klammern im vorletzten Beispiel nicht notwendig sind. Die Erfahrung zeigt, dass die realen Datenbankmanagementsysteme das durchaus anders sehen können. Sie sollten also auch hier die Klammern setzen. Es erspart Probleme und erhöht die Lesbarkeit der SELECT-Anweisung.

Zusammenfassung

Sie können in einer SELECT-Anweisung mehrere JOIN-Verknüpfungen verwenden. Dadurch entstehen ganze Netze von Tabellen, die schrittweise zu virtuellen Tabellen für die Auswertung zusammengefasst werden. Der INNER JOIN erfordert für jede Verbindung zweier Tabellen die Angabe, über welchen Fremdschlüssel und Primärschlüssel die Verbindung erfolgen soll:

(ON tabelle1.Fremdschlüssel = tabelle2.Primärschlüssel).

Eine Klammerung der JOIN-Verknüpfungen ist sinnvoll. Letztlich sind auch die Anforderungen des konkreten SQL-Interpreters des Datenbankmanagementsystems zu beachten. Nicht alle Varianten funktionieren mit allen Systemen.

Übungen

Übungen zur SELECT-Anweisung mit INNER JOIN über mehr als zwei Tabellen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Erstellen Sie eine eindeutige alphabetisch sortierte Liste aller Kursteilnehmer mit Familienname und Vorname, die an einem Kurs zum Thema „Access" teilnehmen. (Ü4.5.2.1)
2. Ermitteln Sie für alle Dozenten ihren Familiennamen, Vornamen, Stundensatz und die Gebühr der Kurse, die sie zurzeit betreuen. (Ü4.5.2.2)

3. Ermitteln Sie zu jedem Kursthema, welche Personen als Dozent welche Kurse zu diesem Thema leiten. Geben Sie das `Kursthema`, die `Kurskennung` sowie Familienname und Vorname der Dozenten aus. Sortieren Sie die Liste alphabetisch nach Dozentennamen. Qualifizieren Sie alle Angaben. (Ü4.5.2.3)
4. Ermitteln Sie alle Personen, die zugleich Kursteilnehmer in einem Kurs und Dozent in einem anderen Kurs sind. (Ü4.5.2.4)

4.5.3 Varianten des INNER JOIN

Der `INNER JOIN`, wie wir ihn bisher vorgestellt haben, wird auch als *Condition-JOIN* bezeichnet, weil eine Bedingung (`ON ...`) für seinen Aufbau verwendet wird. Diese Bedingung (Condition) gibt die Vorschrift an, nach der die Datensätze der Tabellen miteinander verbunden werden sollen. Es gibt aber noch einige weitere Varianten des `INNER JOIN`, die im SQL-Standard definiert sind und in der Praxis – vorsichtig eingesetzt – in bestimmten Situationen vorteilhaft sein können. Dazu gehören insbesondere der *USING JOIN* und der *NATURAL JOIN*. Sie verwenden letztlich auch Bedingungen, um die Tabellen zu kombinieren, allerdings nur implizit.

Wir wollen diese Alternativen zum Condition-`JOIN` jetzt verwenden und können deren spezielle Vor- und Nachteile abwägen.

JOIN mit USING

Der `INNER JOIN` mit `USING` macht sich die Tatsache zunutze, dass sehr oft das Fremdschlüsselfeld der einen Tabelle eines `JOIN` *denselben* Namen hat wie das Primärschlüsselfeld der anderen Tabelle. In diesen Fällen beruht der `JOIN` im Prinzip auf „einem“ Feld. Die `USING`-Variante des `JOIN` macht sich dies zunutze, indem es reicht, den Namen des gemeinsamen Feldes zu erwähnen:

FROM *tabellenname1 INNER JOIN tabellenname2 USING (feldname);*

Beispiel

Soll beispielsweise eine Liste der Kurse mit ihren Dozenten erstellt werden, so kann der `JOIN` über das Feld `DID` erfolgen, das in den beiden Tabellen `tbKurs` und `tbDozent` vorhanden ist. Würden Sie dies als Condition-`JOIN` formulieren, erhalten Sie etwa das Ergebnis in Listing 4.30.

Listing 4.30
Lösung mit Condition-JOIN

```
SELECT d.PID, d.Stundensatz, k.KID, k.Kurskennung, k.Kursbeginn,
       k.Kursende
FROM tbDozent d INNER JOIN tbKurs k ON (d.DID = k.DID);
```

Formulieren Sie dasselbe Problem als `USING JOIN`, erhalten Sie das Ergebnis in Listing 4.31.

Listing 4.31
Lösung mit USING JOIN

```
SELECT d.PID, d.Stundensatz, k.KID, k.Kurskennung, k.Kursbeginn,
       k.Kursende
FROM tbDozent d INNER JOIN tbKurs k USING (DID);
```

Das Ergebnis wäre in beiden Fällen die in Abbildung 4.28 gezeigte Tabelle. Leider unterstützen viele Systeme wie MS Access, openBase oder Firebird diese Syntax aber zumindest zurzeit noch nicht.

Abbildung 4.28
Ergebnis des USING JOIN

PID	Stundensatz	KID	Kurskennung	Kursbeginn	Kursende
2	17,00 €	CE20	Celle20-Word	09.07.2007	13.07.2007
2	17,00 €	CE23	Celle23-Access	06.08.2007	10.08.2007
2	17,00 €	H89	Hannover89-Access	09.07.2007	13.07.2007
2	17,00 €	CE24	Celle24-Access	13.08.2007	17.08.2007
26	14,00 €	CE17	Celle17-Word	23.04.2007	27.04.2007
17	15,00 €	H90	Hannover90-Windows	16.07.2007	18.07.2007
17	15,00 €	H91	Hannover91-Windows	23.07.2007	25.07.2007

NATURAL JOIN

Der `USING JOIN` ist letztlich eine verkürzte Schreibweise für die klassische Variante des CONDITION `JOIN`. Noch kürzer ist die Schreibweise des `NATURAL JOIN`. Dieser „natürliche“ `JOIN` beruht ebenfalls auf gleichnamigen Datenfeldern in den beiden beteiligten Tabellen. Er geht insofern noch einen Schritt weiter als der USING JOIN, als dass alle gleichnamigen Felder als Teil des `JOIN` aufgefasst werden. Es wird also überhaupt kein Datenfeldname mehr angegeben, sondern der SQL-Interpreter überprüft alle Spalten auf mögliche Gleichnamigkeit und erstellt implizit eine Bedingung für jede Kombination gleichnamiger Felder.

FROM *tabellenname1 NATURAL JOIN tabellenname2;*

Mit der Syntax des `NATURAL JOIN` erhält man

Listing 4.32
NATURAL JOIN

```
SELECT d.PID, d.Stundensatz, k.KID, k.Kurskennung, k.Kursbeginn,
       k.Kursende
FROM tbDozent d NATURAL JOIN tbKurs k;
```

Bitte beachten Sie, dass MS Access, openBase und Firebird hier wiederum keine Unterstützung anbieten. In MySQL müssen Sie mit Alias arbeiten, damit der `NATURAL JOIN` funktioniert.

Tipp

Die `USING`-Syntax ist eine eingängige und einfache Syntax, die allerdings nur relativ wenig Schreibarbeit spart. Problematisch ist, dass der `USING JOIN` nur von einigen Datenbankmanagementsystemen unterstützt wird. Sollten Sie planen, verschiedene Datenbankmanagementsysteme zu verwenden, sollten Sie lieber die Syntax mit `ON` verwenden, da diese durchgängig unterstützt wird.

Der `NATURAL JOIN` ist darüber hinaus im produktiven Betrieb generell mit großer Vorsicht zu handhaben. Hier führen Änderungen der Feldnamen zu einer impliziten Änderung des `JOIN`, sei es, dass Feldnamen, die vorher gleich waren, nicht mehr gleich sind, sei es, dass andere Feldnamen plötzlich gleich lauten. Gerade wenn die Ergebnisse der `SELECT`-Anweisung noch von anderen Programmen weiterverwendet werden, kann dies zu schwer aufzuspürenden Fehlern führen.

Übungen

Übungen zur SELECT-Anweisung mit USING JOIN und NATURAL JOIN

Erstellen Sie für die folgenden Aufgaben soweit möglich jeweils eine `SELECT`-Anweisung mit `USING JOIN` und mit `NATURAL JOIN` (nur MySQL und Oracle).

1. Erstellen Sie eine Liste aller Kurse mit Kennung und Kursthema, der tatsächlichen und der geplanten Kursdauer. (Ü4.5.3.1)

2. Erstellen Sie eine alphabetische Liste aller Dozenten mit `Familienname`, `Vorname`, `PLZ`, `Ort` und `Stundensatz` (siehe Ü4.5.1.2). (Ü4.5.3.2)
3. Erstellen Sie eine eindeutige alphabetisch sortierte Liste aller Kursteilnehmer mit `Familienname` und `Vorname`, die an einem Kurs zum Thema „Access“ teilnehmen (siehe Ü4.5.2.1). (Ü4.5.3.3)
4. Ermitteln Sie für alle Dozenten ihren `Familiennamen`, `Vornamen`, `Stundensatz` und die `Gebühr` der Kurse, die sie zurzeit betreuen (siehe Ü4.5.2.2). (Ü4.5.3.4).

4.5.4 Non-Equi-JOIN

NON-EQUI-JOIN

Wir bleiben noch im Bereich des `INNER JOIN`, verallgemeinern diesen aber hinsichtlich des Vergleichsoperators für die beteiligten Felder. Bisher hat jeder `JOIN` immer auf der Gleichheit der Werte in zwei Feldern beruht. Aufgrund der Gleichsetzung der Werte zweier Felder spricht man auch von einem *Equi-JOIN*, also einem `JOIN`, der auf Äquivalenz, also Gleichheit der Werte zweier Datenfelder, beruht. Dies ist die bei Weitem üblichste Art, einen `JOIN` zu bilden. Es gibt allerdings auch Fälle, in denen es nicht darauf ankommt, die Werte zweier Felder mit dem Gleichheitsoperator = zu verbinden, sondern beispielsweise mit einem < oder >. Man spricht auch von einem *Non-Equi-JOIN*.

Wir formulieren jetzt also `JOIN`-Anweisungen, die nicht auf dem Gleichheitsoperator beruhen, und vergleichen damit Daten aus verschiedenen Tabellen über einen `JOIN`.

Beispiel

Sollen beispielsweise alle Kurse ermittelt werden, deren tatsächliche Kursdauer kleiner ist als die im Rahmen der Beschreibung des Kursthemas geplante Dauer, so kann dies mit folgender `SELECT`-Anweisung geschehen:

Listing 4.33
Non-Equi-JOIN

```
SELECT
   k.KID, k.Kurskennung, k.KursdauerStunden,
   kt.Kursthema, kt.DauerPlan
FROM tbKursthema kt INNER JOIN tbKurs k
   ON(k.KTHID = kt.KTHID)
   AND (k.KursdauerStunden < kt.DauerPlan);
```

Das Ergebnis ist in Abbildung 4.29 dargestellt.

KID	Kurskennung	KursdauerStunden	Kursthema	DauerPlan
H90	Hannover90-Windows	18	Einstieg mit Windows	20
H91	Hannover91-Windows	18	Windows für Fortgeschrittene	20
CE23	Celle23-Access	36	Access I	40
H89	Hannover89-Access	32	Access I	40

Abbildung 4.29
Ergebnis des Vergleichs tatsächlicher und geplanter Stunden

Neu ist der Abschnitt

```
AND (k.KursdauerStunden < kt.DauerPlan)
```

der dafür sorgt, dass nur solche Datensätze in der Ergebnismenge erzeugt werden, bei denen die tatsächliche Kursdauer in `k.KursdauerStunden` kleiner als die geplante Kursdauer in `kt.DauerPlan` ist.

Der Vergleich (k.KTHID = kt.KTHID) ist zusätzlich notwendig, damit nur solche Kurse und Kursthemen miteinander in Beziehung gesetzt werden, die einander entsprechen. Fehlte diese Angabe, würden zusätzlich alle Kombinationen ausgegeben, bei denen ein Kurs mit einer kleineren Kursdauer als irgendein anderes Kursthema existiert.

Nutzung der WHERE-Klausel

Ein Non-Equi-JOIN lässt sich in manchen Fällen durch einen Equi-JOIN und eine WHERE-Klausel ersetzen. Der Programmtext in Listing 4.34 zeigt die umformulierte Anweisung mit einem Non-Equi-JOIN in der WHERE-Klausel.

Listing 4.34 Non-Equi-JOIN in der WHERE-Klausel

```
SELECT
   k.KID, k.Kurskennung, k.KursdauerStunden,
   kt.Kursthema, kt.DauerPlan
FROM tbKursthema kt INNER JOIN tbKurs k
   ON(k.KTHID = kt.KTHID)
WHERE (k.KursdauerStunden < kt.DauerPlan);
```

Zusammenfassung

Neben dem Gleichheitsoperator können auch weitere Operatoren für einen JOIN verwendet werden. Insbesondere <, <=, >, >= und != können genutzt werden, um die Felder von Tabellen miteinander über einen JOIN in Verbindung zu setzen. Der Non-Equi-JOIN wird oft nicht in der FROM-, sondern in der alten Syntax in der WHERE-Klausel verwendet.

Übungen

Übungen zur SELECT-Anweisung mit Non-Equi-JOIN

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung

1. Ermitteln Sie die Kursteilnehmer mit KID, bei denen der bezahlte Kursbeitrag (unabhängig vom Rabatt) kleiner ist als die Kursgebühr des besuchten Kurses. Zeigen Sie sowohl die gezahlte Gebühr als auch die Kursgebühr an. (Ü4.5.4.1)
2. Ermitteln Sie die Kurskennungen aller Kurse, die der Dozent Peter Weiss mit der PLZ 30529 nicht betreut. (Ü4.5.4.2)

4.5.5 OUTER JOIN

Der bisher betrachtete innere Verbund (INNER JOIN) in seinen verschiedenen Formen lässt die Kombination von Datensätzen aus zwei Tabellen zu, die in irgendeiner Form einander entsprechen. Es wurden immer die Werte zweier Felder miteinander verglichen und Datensätze mit passenden Werten miteinander kombiniert. Anders stellt sich die Situation dar, wenn in einem der am JOIN beteiligten Felder in einem Datensatz ein passender Wert in der anderen Tabelle fehlt.

Beispiel

Soll der Einsatz aller Dozenten in Kursen analysiert werden, so lässt sich dies durch einen JOIN zwischen tbDozent und tbKurs erreichen. Was ist aber mit den Dozenten, für die zurzeit kein Einsatz geplant ist? Für sie existiert kein passender Satz in der Kurstabelle, daher fallen sie bei einem INNER JOIN wie in Listing 4.35 „unter den Tisch“.

Listing 4.35
INNER JOIN zwischen Dozent und Kurs

```
SELECT
   tbPerson.Familienname, tbPerson.Vorname, tbDozent.Titel,
   tbDozent.Qualifikationen, tbKurs.Kurskennung
FROM (tbDozent INNER JOIN tbKurs
   ON (tbDozent.DID = tbKurs.DID))
   INNER JOIN tbPerson ON (tbDozent.PID = tbPerson.PID)
ORDER BY tbPerson.Familienname, tbPerson.Vorname;
```

Abbildung 4.30
Ergebnis mit dem INNER JOIN: eingesetzte Dozenten

Familienname	Vorname	Titel	Qualifikationen	Kurskennung
Bucz	Susanne	Industriekaufmann	Windows, Rechnungswesen	Hannover90-Windows
Bucz	Susanne	Industriekaufmann	Windows, Rechnungswesen	Hannover91-Windows
Cromberg	Jörg	Mathematiker	Windows, Word	Celle17-Word
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Hannover89-Access
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Celle24-Access
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Celle23-Access
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Celle20-Word

Um auch die Datensätze einer Tabelle zu erhalten, die in einer anderen Tabelle keine Entsprechung finden, benötigen wir den `OUTER JOIN`.

Beispiel

Es sollen beispielsweise alle Dozenten mit ihren in der Datenbank abgelegten Kurseinsätzen angegeben werden. Als Besonderheit wollen wir auch diejenigen Dozenten auflisten, die zurzeit nicht in Kursen eingeplant sind. Bei den Dozenten wird der Familienname und Name angegeben, nach denen auch sortiert wird.

Listing 4.36
OUTER JOIN zwischen Dozent und Kurs

```
SELECT
   tbPerson.Familienname, tbPerson.Vorname, tbDozent.Titel,
   tbDozent.Qualifikationen, tbKurs.Kurskennung
FROM (tbDozent LEFT OUTER JOIN tbKurs
   ON (tbDozent.DID = tbKurs.DID))
   INNER JOIN tbPerson ON (tbDozent.PID = tbPerson.PID)
ORDER BY tbPerson.Familienname, tbPerson.Vorname;
```

Neu ist die Angabe

```
tbDozent LEFT OUTER JOIN tbKurs
ON (tbDozent.DID = tbKurs.DID)
```

die dazu führt, dass aus der linken Tabelle, also `tbDozent`, alle Datensätze Teil der Ergebnismenge werden, egal ob in der Tabelle `tbKurs` ein Kurs vorhanden ist oder nicht. Entsprechend erkennen Sie im Ergebnis dieser Abfrage in Abbildung 4.31, dass den beiden Dozenten Dieter Schlachter und Karin Weiss keine Kurse zugeordnet sind.

LEFT OUTER JOIN

Wäre statt eines `LEFT OUTER JOIN`hier ein `INNER JOIN` verwendet worden, hätten beide Datensätze im Ergebnis komplett gefehlt (siehe Abbildung 4.30).

Abbildung 4.31
Ergebnis des OUTER JOIN

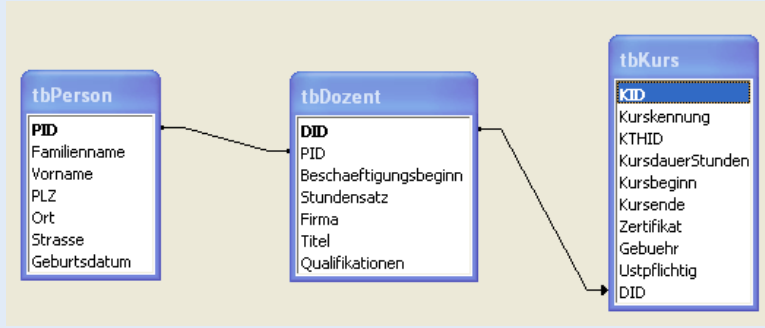
Familienname	Vorname	Titel	Qualifikationen	Kurskennung
Bucz	Susanne	Industriekaufmann	Windows, Rechnungswesen	Hannover91-Windows
Bucz	Susanne	Industriekaufmann	Windows, Rechnungswesen	Hannover90-Windows
Cromberg	Jörg	Mathematiker	Windows, Word	Celle17-Word
Schlachter	Dieter	IT-Kaufmann	Office, Datenbanken, Netzwerke	
Weiss	Karin	Meister	CNC-Steuerungen	
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Celle24-Access
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Hannover89-Access
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Celle23-Access
Weiss	Peter	Informatiker	Word, Windows, Datenbanken	Celle20-Word

Der OUTER JOIN in MS Access

In MS Access ist ein `OUTER JOIN` möglich. MS Access erforderte in der Syntax bis Access 2000 nur, dass die Angabe `OUTER` weggelassen wird. Es muss hier also `LEFT JOIN` statt `LEFT OUTER JOIN` heißen. Entsprechendes gilt für den `RIGHT OUTER JOIN`, der in MS Access zum `RIGHT JOIN` wird. Ab MS Access 2003 funktioniert die Standardsyntax.

Außerdem bietet MS Access die Möglichkeit, einen `OUTER JOIN` direkt über die grafische Oberfläche zu bestimmen und anzuzeigen. Die zeigt, dass man einen `OUTER JOIN` als gerichteten `JOIN` auffassen kann. Bei einem `LEFT OUTER JOIN` wird von der linken Tabelle ausgegangen, aus der alle Datensätze genommen werden. Aus der rechten Tabelle werden dann wie gewohnt passende Datensätze ergänzt. Fehlen diese Datensätze, bleiben die entsprechenden Feldwerte in der Ergebnismenge leer. Die Datensätze der „linken" Tabelle werden aber vollständig in die Ergebnismenge übernommen.

Abbildung 4.32
LEFT OUTER JOIN zwischen tbDozent und tbKurs



Darüber hinaus bietet MS Access eine zusätzliche Hilfe beim Aufbau eines `OUTER JOIN`. Gehen Sie dazu in den oberen Teil des Abfragefensters, der die Verbindungen zwischen den Tabellen visualisiert. Hier können Sie durch einen einfachen Klick auf die Verbindung diese zunächst aktivieren, sie wird hervorgehoben. Jetzt erhalten Sie durch einen rechten Mausklick auf diese Verbindung ein kleines Menü, in dem Sie die VERKNÜPFUNGSEIGENSCHAFTEN aufrufen können. Alternativ können Sie auch über den Menübefehl ANSICHT/VERKNÜPFUNGSEIGENSCHAFTEN gehen.

Sie erhalten ein neues Fenster, wie in dargestellt. Wenn Sie jetzt den VERKNÜPFUNGSTYP wählen, erhalten Sie ein neues Fenster. Hier finden Sie die Standardeinstellungen

1. für einen `INNER JOIN`,
2. für einen `LEFT OUTER JOIN`,
3. für einen `RIGHT OUTER JOIN`.

MS Access erläutert die Bedeutung der einzelnen Beziehungen am Beispiel der beteiligten Tabellen. Nehmen Sie sich die Zeit, die Beschreibung kurz zu lesen, ist diese zumeist auch recht verständlich.

Beziehungen bearbeiten
Tabelle/Abfrage:
Verwandte Tabelle/Abfrage:
tbDozent
tbKurs
DID
DID
OK
Abbrechen
Verknüpfungstyp...
Neue erstellen...
Mit referentieller Integrität
Aktualisieren
Löschweite
Beziehungstyp:
Verknüpfungseigenschaften
1: Beinhaltet nur die Datensätze, bei denen die Inhalte der verknüpften Felder beider Tabellen gleich sind.
2: Beinhaltet ALLE Datensätze aus 'tbDozent' und nur die Datensätze aus 'tbKurs', bei denen die Inhalte der verknüpften Felder beider Tabellen gleich sind.
3: Beinhaltet ALLE Datensätze aus 'tbKurs' und nur die Datensätze aus 'tbDozent', bei denen die Inhalte der verknüpften Felder beider Tabellen gleich sind.
OK
Abbrechen

Abbildung 4.33
Einstellung für einen LEFT OUTER JOIN

RIGHT OUTER JOIN

Neben dem `LEFT OUTER JOIN` gibt es natürlich auch die umgekehrte Variante, den `RIGHT OUTER JOIN`. Dabei werden alle Datensätze aus der rechten Tabelle übernommen und – soweit vorhanden – die passenden Datensätze aus der linken Tabelle dazu kombiniert.

Ein `OUTER JOIN` lässt sich mit einem anderen `OUTER JOIN` oder `INNER JOIN` kombinieren. Dabei treten allerdings in der Praxis immer Probleme auf, wenn die „offene“ Seite eines `JOIN`, also die rechte Seite eines `LEFT OUTER JOIN` oder die linke Seite eines `RIGHT OUTER JOIN`, in einem weiteren `JOIN` verwendet werden soll. Die Ursache ist in vielen Fällen in den sogenannten `NULL`-Werten zu suchen. `NULL` steht dabei für das Fehlen jeden Wertes, also das, was den `OUTER JOIN` ausmacht. Als Faustregel bei der Kombination von `INNER JOIN` und `OUTER JOIN` gilt, dass zunächst so viele `INNER JOIN` bearbeitet werden wie benötigt, nach einem `OUTER JOIN` sollte aber kein `INNER JOIN` mehr folgen.

Info

Der SQL-Standard SQL99 sieht auch einen `FULL OUTER JOIN` vor, der der Kombination des `LEFT OUTER JOIN`mit dem `RIGHT OUTER JOIN`entspricht, also alle jeweiligen Datensätze ohne passenden Datensatz der anderen Tabelle beinhaltet. Dies würde dann in unserem Beispiel nicht nur die Dozenten ohne Kurse, sondern auch die Kurse ohne Dozenten, die es hoffentlich nicht gibt, im Ergebnis liefern.

Zusammenfassung

Mit einem `OUTER JOIN` werden bei einem `JOIN` auch Datensätze ausgegeben, zu denen keine „passenden" Datensätze in der anderen Tabelle existieren.

Tabelle 4.2 Übersicht über das Ergebnis eines OUTER JOIN

JOIN	Ergebnis
`tabelle1 LEFT OUTER JOIN tabelle2`	Alle Datensätze aus `tabelle1`
`tabelle1 RIGHT OUTER JOIN tabelle2`	Alle Datensätze aus `tabelle2`
`tabelle1 FULL OUTER JOIN tabelle2`	Alle Datensätze aus `tabelle1` und `tabelle2`

Übungen zur SELECT-Anweisung mit einem OUTER JOIN

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Ermitteln Sie für alle Kursthemen in `tbKursthema`, welche Kurse dazu in `tbKurs` tatsächlich angeboten werden. Berücksichtigen Sie auch alle Kursthemen, die zurzeit nicht angeboten werden. Es sollen jeweils das Kursthema, die Kurskennung sowie Kursbeginn und Kursende angezeigt werden. (Ü4.5.5.1)
2. Ermitteln Sie für alle Personen, ob und an welchen Kursen (`KID` reicht) sie teilgenommen haben. Die Personen sollen nach Familienname und Vorname alphabetisch sortiert werden. Die `PID` soll für jede Person angegeben werden. Für die Kursteilnehmer sollen die `KID` und die `Zahlweise` angegeben werden. Es sollen auch alle Personen aufgelistet werden, die zurzeit keine Kursteilnehmer sind. (Ü4.5.5.2)
3. Ermitteln Sie für alle Personen, ob und an welchen Kursen (`KID` reicht) sie teilnehmen oder ob sie als Dozent tätig sind. Die `PID` soll mit angezeigt werden. Für die Kursteilnehmer sollen die `KID` und die `Zahlweise` angegeben werden. Für die Dozenten sollen die `DID` und die Qualifikation angezeigt werden. Es sollen auch alle Personen aufgelistet werden, die zurzeit weder Kursteilnehmer noch Dozent sind. (Ü4.5.5.3)

4.5.6 CROSS JOIN

Ein `JOIN` stellt grundsätzlich die Verbindung zweier Tabellen dar. Wir haben jetzt verschiedene Formen von Bedingungen gesehen, mit denen festgelegt wird, welche Datensätze der beiden Tabellen miteinander zu verbinden sind. Der `INNER JOIN` erforderte zumeist die Gleichheit, zumindest aber die Ver-

gleichbarkeit eines oder mehrerer Felder. Der `OUTER JOIN` akzeptiert auch das Fehlen eines passenden Datensatzes auf einer oder beiden Seiten.

Kartesisches Produkt

Was passiert aber, wenn zwei Tabellen miteinander verbunden werden, ohne überhaupt irgendeine Bedingung für die Kombination zweier Datensätze anzugeben? Da ist der SQL-Interpreter rigoros. Wenn keine einschränkende Bedingung für die Kombination zweier Datensätze vergeben wird, werden einfach alle Datensätze einer Tabelle mit allen Datensätzen der anderen Tabelle kombiniert. Mathematisch wird dies als kartesisches Produkt bezeichnet. In SQL heißt es `CROSS JOIN`. In der Praxis ist dies nahezu immer ein Problem. Normalerweise ist es nicht sinnvoll, wahllos alle Datensätze miteinander zu kombinieren. Es macht schließlich wenig Sinn, beispielsweise einen Kunden auch mit allen Aufträgen der anderen Kunden zu kombinieren. Trotzdem ist die entsprechende Syntax im SQL-Standard definiert worden.

CROSS JOIN-Syntax

FROM *tabellename1 CROSS JOIN tabellenname2*

Den `CROSS JOIN` gab es de facto bereits vor dessen Definition. In der SQL89-Syntax ergibt sich ein `CROSS JOIN` immer dann (zumeist ungewollt), wenn man vergisst, zwei Tabellen miteinander in der `WHERE`-Klausel zu verbinden wie etwa in Listing 4.37.

Listing 4.37 CROSS JOIN durch fehlende Verknüpfungsbedingung

```
SELECT
   k.KID, k.GezahlterBetrag, p.Familienname,
   p.Vorname, p.PLZ, p.Ort, p.Strasse
FROM tbPerson p, tbKursbesuche k
WHERE (k.KID = 'CE23') AND (k.GezahlterBetrag < 250)
ORDER BY p.Familienname, p.Vorname;
```

Es fehlt in der Anweisung eine Verbindung der beiden Tabellen, also beispielsweise ein

```
(p.PID = k.KTID)
```

und schon werden (alle nicht über die `WHERE`-Klausel herausgefilterten) Datensätze der Tabelle `tbKursbesuche` mit allen Datensätzen der Tabelle `tbPerson` kombiniert. Bei großen Datenbanken hat man dann in der Regel riesige Ergebnismengen mit entsprechenden Laufzeiten und ohne inhaltliche Bedeutung. Trotzdem gibt es Spezialfälle, in denen ein `CROSS JOIN` tatsächlich beabsichtigt werden kann.

Beispiel

Wenn Sie einmal große Datenmengen erzeugen wollen, um beispielsweise die Auswirkung auf Ihre Datenbankperformance zu testen oder um Optimierungen wie Indizes zu testen, können Sie dies recht schnell mit einem `CROSS JOIN` erreichen.

Im folgenden Beispiel finden Sie vier Tabellen, `CJVorname`, `CJFamilienname`, `CJOrt` und `CJStrasse`, die jeweils nur aus einem Feld bestehen. In den Tabellen sind zehn Vornamen, zehn Familiennamen, zehn Orte und zehn Straßen eingetragen. Mit dem einfachen `CROSS JOIN`, hier in der „alten Syntax“, die in allen Datenbanksystemen funktioniert,

Listing 4.38
Beispiel – CROSS JOIN für die Adressen

```
SELECT
   CJVorname.Vorname,
   CJFamilienname.Familienname,
   CJOrt.Ort,
   CJStrasse.Strasse
FROM CJFamilienname, CJOrt, CJStrasse, CJVorname;
```

werden alle Datensätze aller Tabellen miteinander kombiniert. Also werden zehn Vornamen mit zehn Familiennamen kombiniert, was 100 Datensätze ergibt.

Dann werden diese 100 Datensätze mit den zehn Datensätzen der Ortstabelle kombiniert, was 1.000 Datensätze ergibt.

Diese 1.000 Datensätze werden dann mit den zehn Datensätzen der Straßentabelle kombiniert, woraus immerhin 10.000 Datensätze resultieren.

Mit der neuen `CROSS JOIN`-Syntax lautet die `SELECT`-Anweisung entsprechend:

Listing 4.39
CROSS JOIN mit neuer Syntax

```
SELECT
   CJVorname.Vorname,
   CJFamilienname.Familienname,
   CJOrt.Ort,
   CJStrasse.Strasse
FROM CJFamilienname
   CROSS JOIN CJOrt
   CROSS JOIN CJStrasse
   CROSS JOIN CJVorname;
```

Der `CROSS JOIN` ist in openBase und MS Access nicht definiert und daher nur in der alten Syntax einsetzbar.

Mit einer `CREATE`-Anweisung, die das `SELECT` als Unterabfrage verwendet (siehe Kapitel 10), kann daraus auch schnell eine neue Tabelle erzeugt werden. Übrigens, bevor Sie dies als MS Access-Nutzer probieren oder gar die Anzahl noch einmal mit 10 multiplizieren wollen, sollten Sie dringend eine Sicherheitskopie Ihrer Datenbank anlegen, wenn Sie noch Freude an der Datenbank haben wollen.

Listing 4.40
Erzeugung einer neuen Tabelle mit Unterabfrage durch CROSS JOIN

```
CREATE TABLE CJAdresse
(
   SELECT CJVorname.Vorname, CJFamilienname.Familienname,
   CJOrt.Ort,CJStrasse.Strasse
   FROM CJFamilienname, CJOrt, CJStrasse, CJVorname
);
```

Das Ergebnis ist hier eine neue Tabelle mit 10.000 Datensätzen, die alle möglichen Kombinationen von Vorname, Familienname, Ort und Straße beinhalten.

Übungen zur SELECT-Anweisung mit einem CROSS JOIN

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Ermitteln Sie alle möglichen Kombinationen von Dozenten (mit `DID` und `Stundensatz`) und Kursthemen, wenn theoretisch jeder Dozent jeden Kurs halten kann. Interpretieren Sie das Ergebnis in der Praxis. (Ü4.5.6.1)

2. Erweitern Sie die Abfrage aus Ü4.5.6.1 so, dass neben der `DID` auch der richtige `Familienname` und `Vorname` des Dozenten angezeigt werden. Dabei werden ein `INNER JOIN` und ein `CROSS JOIN` kombiniert. (Ü4.5.6.2)

4.5.7 JOIN über mehrere Felder

Normalerweise hat jede Tabelle in einer Datenbank nur ein Primärschlüsselfeld. Dies ist aber nicht zwingend. Eine Tabelle kann auch über zwei oder mehr Felder verfügen, die zusammen ihren Primärschlüssel bilden.

Beispiel

In bestimmten Fällen ist die Verwendung mehrerer Primärschlüsselfelder durchaus sinnvoll – beispielsweise in Warenwirtschaftssystemen oder ERP-Systemen, die mandantenfähig sein sollen. Die Mandantenfähigkeit beschreibt die Möglichkeit, in einer Datenbank die Daten von mehr als einem Unternehmen zu verwenden. Dazu wird eine sogenannte Mandantennummer verwendet, die zu allen Tabellen hinzugefügt wird, die mandantenfähig sein sollen. Der Vorteil ist, dass verbundene oder eng kooperierende Unternehmen, die rechtlich selbstständig sind, ihre Daten in derselben Datenbank führen können. Die Datensätze werden mithilfe der Mandantennummer unterschieden, indem jedes Unternehmen seine eigene Mandantennummer erhält.

Sollen beispielsweise die Kunden nach den Mandanten getrennt werden, wird in einer Tabelle `tbKunde` neben dem ursprünglichen Primärschlüsselfeld `kdnr` (Kundennummer) eine `mnr` (Mandantennummer) aufgenommen. Damit kann von verschiedenen Unternehmen dieselbe Kundennummer vergeben werden. Die Kundenstämme können mithilfe der unterschiedlichen Mandantennummer unterschieden werden. Es existieren bei zwei Mandanten sozusagen zwei parallele Datenbestände, für jeden Mandanten ein eigener Bestand. Dies erfordert dann aber natürlich auch, dass andere Tabellen, die sich auf diese Tabelle beziehen, zwei Fremdschlüsselfelder verwenden, eines für die `kdnr`, eines für die `mnr`. Der Bezug zu dieser Tabelle mithilfe eines solchen Fremdschlüssels erfordert dann einen `JOIN` mit mehreren Feldern (siehe Abbildung 4.34).

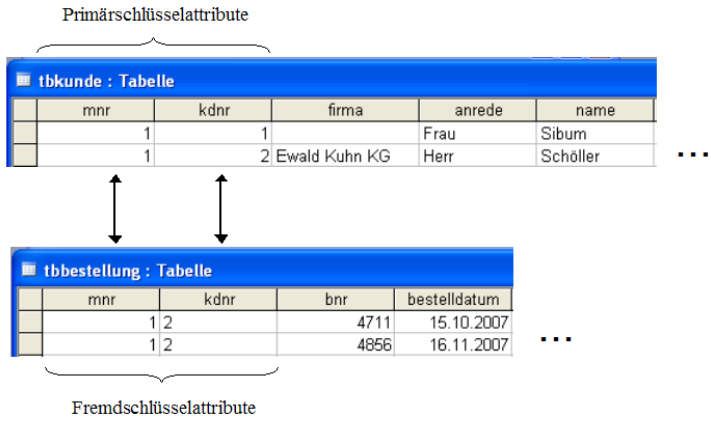


Abbildung 4.34
Mandantenfähige Verbindung der Kunden und Bestellungen

Ein JOIN muss dann alle betroffenen Felder, hier die kdnr und die mnr, berücksichtigen.

Listing 4.41
JOIN über zwei Feldpaare

```
SELECT
   k.mnr, k.kdnr, k.firma, k.anrede, k.name,
   b.bnr, b.bestelldatum
FROM
   tbkunde k INNER JOIN tbBestellung b
   ON (k.mnr = b.mnr) AND (k.kdnr = b.kdnr);
```

Diese Abfrage berücksichtigt beide Primärschlüsselfelder und ihre entsprechenden Fremdschlüsselfelder.

Sie können auch die anderen JOIN-Arten im Zusammenhang mit Verbindungen über zwei oder mehr Feldpaare entsprechend verwenden. So lassen sich in der USING-Klausel mehrere Feldnamen aufzählen:

USING (*mnr, kdnr)*

Der NATURAL JOIN, der auf Feldnamengleichheit beruht, würde hier sogar automatisch beide Felder berücksichtigen, da beide jeweils passende gleiche Namen haben. Weitere Beispiele und Übungen finden Sie im Rahmen der Beispiele mit der Artikeldatenbank, die für komplexere Situationen genutzt wird, siehe hierzu auch Kapitel 9.

4.6 Die GROUP BY-Klausel

Datenbanken enthalten – im Gegensatz zu unseren Testbeispielen – zumeist große Mengen von Datensätzen. Meistens ist man nur an wenigen Datensätzen interessiert und filtert die benötigten Datensätze über die WHERE-Klausel heraus. Oft möchte man aber auch den Überblick über Gruppen gleichartiger Datensätze haben, beispielsweise Stammkunden, alle Autos von VW oder alle Teilnehmer bestimmter Kurse. In diesen Fällen werden die Datensätze mit gleichen Eigenschaften über eine GROUP BY-Klausel vorsortiert.

Es geht also darum, Daten zu verdichten, um beispielsweise Stückzahlen, Summen oder Durchschnittswerte zu erhalten. Dazu gruppieren Sie die Datensätze einer SELECT-Anweisung nach einem relevanten Gruppierungsfeld und verdichten die Daten in anderen Feldern, bei denen die Werte in dem Gruppierungsfeld gleich sind.

Beispiel

Sie möchten beispielsweise die Teilnehmer unserer Kurse nach den Kursen gruppieren, an denen sie teilnehmen, und zusätzlich die Anzahl der Teilnehmer an den verschiedenen Kursen ermitteln. Beginnen wir dazu mit der SQL-Anweisung in Listing 4.42.

Listing 4.42
Liste der besuchten Kurse

```
SELECT kb.KID AS "Anzahl Kursteilnehmer"
FROM tbKursbesuche kb
ORDER BY kb.KID;
```

Das Ergebnis ist in der Abbildung 4.35 zu sehen. Jetzt können Sie natürlich in der Tabelle selbst zählen, wie viele Teilnehmer die einzelnen Kurse haben. Dank der Sortierung geht das sogar relativ einfach. Man kommt schnell auf drei Teilnehmer am Kurs „CE17“ und die acht Teilnehmer des Kurses „CE23“ sind im Normalfall auch noch an den Fingern zweier Hände abzuzählen.

Anzahl Kursteilnehmer
CE17
CE17
CE17
CE23
CE23
CE23
CE23
CE23
CE23
CE23
CE23
CE24
CE24
CE24
CE24
CE24
H90
H90

Abbildung 4.35
Einzeldatensätze der Tabelle Kursteilnehmer

Bei größeren Datenmengen wird dies aber schnell schwierig und die Frage nach einer Unterstützung durch die Datenbank ist sicher nicht als unverschämt abzutun. Hier kann die folgende SQL-Anweisung Abhilfe schaffen:

```
SELECT kb.KID, COUNT(*) AS "Anzahl Kursteilnehmer"
FROM tbKursbesuche AS kb
GROUP BY kb.KID;
```

Listing 4.43
Einstufig gruppierte Liste der besuchten Kurse

KID	Anzahl Kursteilnehmer
CE17	3
CE23	8
CE24	5
H90	2

Abbildung 4.36
Ergebnis der SELECT-Anweisung mit GROUP BY-Klausel

Die SELECT-Anweisung ermittelt das Ergebnis, das Sie in Abbildung 4.36 sehen. Schauen wir uns die neuen Teile in der Anweisung an. Zunächst haben wird die GROUP BY-Klausel

```
GROUP BY kb.KID
```

Gruppendatensatz

die dazu führt, dass die Datensätze so sortiert werden, dass alle Datensätze mit demselben Wert im Datenfeld `kb.KID` hintereinanderstehen. Dann wird aus diesen Datensätzen ein einziger neuer (virtueller) Datensatz erstellt, der auch als *Gruppendatensatz* bezeichnet wird. Ein Gruppendatensatz repräsentiert also immer eine Gruppe von einzelnen Datensätzen, die sonst in mehreren Zeilen getrennt dargestellt werden. Die Gruppe definiert sich, wie sonst auch, über mindestens eine Gemeinsamkeit, hier über einen gleichen Wert in einem bestimmten Datenfeld.

Ein Feld, hier die `KID`, enthält also für alle Einzeldatensätze einer Gruppe denselben Wert. Dies bezeichnet man als Gruppenbildung oder Gruppierung, die durch die `GROUP BY`-Klausel definiert wird:

GROUP BY *feldname* { ,*feldname*}

Für die Gruppierung dürfen nur Felder verwendet werden, die in den durch die `FROM`-Klausel ausgewählten Tabellen vorhanden sind, das heißt, dass die Gruppierungsfelder nicht zwingend auch angezeigt werden müssen.

Gruppierungsfelder

Nachdem eine Gruppierung über die `GROUP BY`-Klausel definiert ist, können jetzt weitere Felder ergänzt werden. Dabei muss für jedes einzelne Feld festgelegt werden, wie die Werte der einzelnen Datensätze zu einem Wert des neuen Gruppendatensatzes zusammengefasst werden. Die einfachste Form ist dabei ein neues Feld, das einfach die Anzahl der Datensätze pro Gruppe zählt. Dafür wird die Funktion `COUNT(feldname)` verwendet. `COUNT(*)` steht für alle Datensätze der Tabelle. Wie gesagt darf das Gruppierungsfeld in der Ergebnismenge fehlen, sodass Sie mit

Listing 4.44
Anzahl der Kursteilnehmer, gruppiert je Kurs

```
SELECT COUNT(*) AS "Anzahl Kursteilnehmer"
FROM tbKursbesuche AS kb
GROUP BY kb.KID;
```

beispielsweise eine Liste mit der Anzahl der Teilnehmer aller Kurse erhalten, ohne dass die Kursnummer in der Ausgabe dabeisteht – der Sinn sei für den Moment dahingestellt.

Durch die Gruppierung wird die Menge der Ergebnisdatensätze immer in eine Teilmenge von Datensätzen zerlegt. In jeder Teilmenge sind die Werte des Gruppierungsfeldes identisch und die Gesamtmenge zerfällt entsprechend der Abbildung 4.37.

Die Gruppierung kann grundsätzlich nach beliebig vielen Feldern erfolgen, wobei die Reihenfolge der Gruppierung entsprechend der Angabe der Feldnamen von links nach rechts durchgeführt wird. Der am weitesten links stehende Feldname legt die erste Aufteilung in Gruppen fest, der nächste Feldname gruppiert innerhalb der Gruppen, die schon gebildet sind. Der dritte dann wiederum innerhalb dieser Gruppen. Bei mehreren Gruppierungsfeldern müssen also in jeder Gruppe die Werte aller Gruppierungsfelder identisch sein.

Mehrstufige Gruppierung

Soll beispielsweise ermittelt werden, wie sich die Struktur der Zahlungsweise innerhalb der Kursteilnehmer der einzelnen Kurse zusammensetzt, so kann zunächst nach der `KID`, also dem Kurs, und dann innerhalb des Kurses nach der Zahlungsweise gruppiert werden:

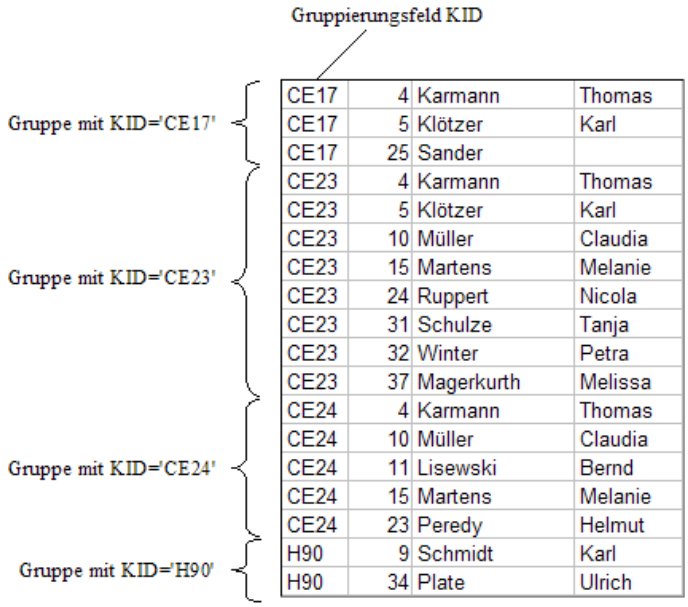


Abbildung 4.37
Gruppierung nach der Kurs-Identifikationsnmmer (KID)

```
SELECT kb.KID, kb.Zahlweise, COUNT(*) AS "Anzahl Zahler"
FROM tbKursbesuche kb
GROUP BY kb.KID, kb.Zahlweise;
```

Listing 4.45
Mehrstufige Gruppierung

Die entstehende Struktur ist in Abbildung 4.38 dargestellt. Es entstehen 10 Gruppen. In jeder der Gruppen sind die Werte beider Gruppierungsfelder in allen Datensätzen gleich.

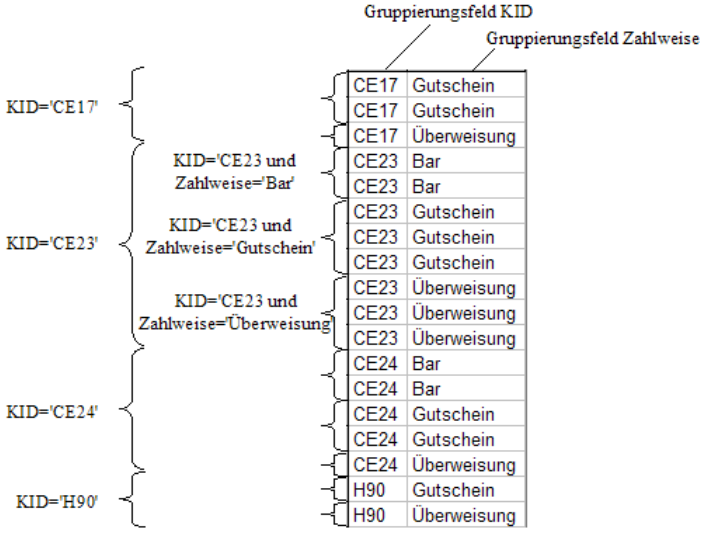


Abbildung 4.38
Zweifache Gruppierung führt zu 10 Gruppen.

Entsprechend können jetzt die Datensätze in den einzelnen Gruppen mit `COUNT(*)` gezählt werden und Sie erhalten das Ergebnis in Abbildung 4.39. Wollen Sie beispielsweise die Werte für den Kurs „CE23“ verifizieren, so können Sie leicht nachzählen, dass es sich um zwei, drei und noch einmal drei Datensätze handelt.

Abbildung 4.39
Ergebnis bei zweifacher Gruppierung

KID	Zahlweise	Anzahl Zahler
CE17	Gutschein	2
CE17	Überweisung	1
CE23	Bar	2
CE23	Gutschein	3
CE23	Überweisung	3
CE24	Bar	2
CE24	Gutschein	2
CE24	Überweisung	1
H90	Gutschein	1
H90	Überweisung	1

Virtueller Datensatz

In einer `SELECT`-Anweisung mit Gruppenbildung wird jeweils aus allen tatsächlichen Datensätzen einer Gruppe ein virtueller Gruppendatensatz gebildet. Dieser Gruppendatensatz enthält alle Felder, die in der `SELECT`-Anweisung vorgegeben sind. Für die Ausgabe muss daher für jedes Feld aus allen Werten des Feldes in allen Datensätzen der Gruppe genau ein Wert ermittelt werden. Mehr als ein Wert je Feld ist auch in dem virtuellen Gruppendatensatz nicht möglich.

Wert des Gruppierungsfeldes

Für das Gruppierungsfeld ist die Wertermittlung einfach. Schließlich ist die Gruppierung gerade so gebildet worden, dass der Wert in dem Feld einer Gruppe überall gleich und somit auch für den Gruppendatensatz sinnvoll ist. Hier würde für die erste Gruppe das Feld `KID` beispielsweise den Wert „CE17“ bekommen. Bei zweifacher Gruppierung gilt dies für beide Gruppierungsfelder, beispielsweise für `KID` „CE17“ und für die `Zahlweise` „Gutschein“. Damit ist klar, welcher Wert für die Gruppierungsfelder ausgegeben wird, wenn sie Bestandteil der Ausgabe sind.

Bei allen anderen Feldern muss aus der Menge der Werte aller Datensätze innerhalb der Gruppe ein Wert gewonnen werden. Dazu werden sogenannte *Aggregatfunktionen* wie `COUNT()` verwendet. Diese Funktionen beinhalten die Anweisung, wie die Einzelwerte zu einem gemeinsamen Wert verdichtet werden sollen. Diese Anweisung ist notwendig, weil im Gegensatz zu den Gruppierungsfeldern die Werte der anderen Felder innerhalb einer Gruppe normalerweise nicht gleich sind. Also gilt die Regel, dass bei einer Gruppierung alle Felder außer den Gruppierungsfeldern eine Aggregatfunktion besitzen müssen.

Aggregatfunktion

Die Aggregatfunktion `COUNT()` beispielsweise bewirkt einfach die Zählung der Datensätze der Gruppe. Das wäre aber doch sehr wenig. Daher gibt es daneben die Möglichkeit, durch andere Aggregatfunktionen die Zusammenfassung (Aggregation) der Werte eines Datenfeldes aller Einzeldatensätze

einer Gruppe durchzuführen. Aggregatfunktionen sind in der Lage, beispielsweise den kleinsten Wert einer Gruppe zu ermitteln (MIN), die Werte zu summieren (SUMME) oder einen Mittelwert (AVG) zu bilden. Dabei wird die Funktion mit dem Feldnamen aufgerufen, also beispielsweise:

- `SUM(Rabatt)` – Summe aller Rabatte je Gruppe
- `AVG(GezahlterBetrag)` – durchschnittlich gezahlter Betrag je Gruppe
- `COUNT(Zahlweise)` – Anzahl (unterschiedlicher) Zahlweisen je Gruppe.

Wesentlich ist wie gesagt, dass für jedes Datenfeld außer den Gruppierungsfeldern eine Aggregatfunktion angegeben wird. Ist nämlich nicht klar, wie für ein Feld ein repräsentativer Gruppenwert entstehen soll, müsste der SQL-Interpreter selbst entscheiden, welchen Wert er nehmen soll. Das ist nicht nur willkürlich, sondern auch sinnlos. Also können keine einfachen Felder im Ergebnis der SELECT-Anweisung auftauchen, sondern nur Gruppierungsfelder und Felder mit Aggregatfunktion.

So kann beispielsweise die Summe der gezahlten Beiträge je Kurs wie folgt ermittelt werden:

Listing 4.46
Je Kurs bezahlte Beträge

```
SELECT KID, SUM(GezahlterBetrag)
FROM tbKursbesuche
GROUP BY KID
ORDER BY 2 DESC;
```

Das Feld `KID` ist das Gruppierungsfeld, es wird also je Kurs ein Gruppendatensatz erzeugt. Dann werden die Werte des Feldes `GezahlterBetrag` je Gruppe summiert. Abschließend werden die ermittelten Datensätze absteigend nach dem summierten Betrag (2. Spalte) sortiert.

Gruppierung mit MS Access

Gruppierung in MS Access und openBase

An dieser Stelle ist es Zeit, kurz auf die Besonderheiten von MS Access und openBase einzugehen. Beide bieten in der grafischen Oberfläche eine besondere Unterstützung für die Gruppierung an, die dann in eine `GROUP BY`-Klausel in SQL umgesetzt wird. Schauen wir uns dabei zunächst das Beispiel an. Beachten Sie für MS Access die eckigen Klammern bei dem Alias.

In der grafischen Oberfläche von MS Access erhalten Sie die Darstellung in Abbildung 4.40. Auffällig ist dabei die Zeile FUNKTION. Diese wird in der Standardeinstellung nicht angeboten. Sie können sie über den Menübefehl ANSICHT/FUNKTIONEN einblenden. Alternativ geht dies auch über die rechte Maustaste im unteren Fensterbereich und durch die Anwahl von FUNKTIONEN im dann eingeblendeten Kontextmenu oder über das Summen-Icon.

Gruppierung festlegen

In der Zeile FUNKTION bestimmen Sie die Gruppierung, indem Sie für die Gruppierungsfelder die Funktion GRUPPIERUNG einstellen. Für alle anderen Felder stellen Sie die Aggregatfunktion – für unser Beispiel SUMME – ein, was dann in ein `SUM` in SQL umgesetzt wird. Beachten Sie bitte, dass die Gruppierungsfelder in der Reihenfolge von links nach rechts in die `GROUP BY`-Klausel übernommen werden. Sie müssen also bei mehreren Gruppierungsfeldern

gegebenenfalls die Reihenfolge der Felder so anpassen, dass die gewünschte Gruppierung entsteht.

Abbildung 4.40
MS Access-Darstellung der GROUP BY-Klausel als Gruppierung

Totale Gruppierung

Eine weitere Besonderheit von MS Access ist, dass sobald die FUNKTIONEN aktiviert sind, grundsätzlich zunächst alle Felder, beginnend von links nach rechts, gruppiert werden. Access bildet also im Standard in jeder Abfrage eine totale Gruppierung. Sie können dies gut nachvollziehen, indem Sie die SELECT-Anweisung

```
SELECT kb.KID, kb.Zahlweise
FROM tbKursbesuche kb
GROUP BY kb.KID, kb.Zahlweise;
```

einmal mit und einmal ohne eingeschaltete FUNKTION-Zeile ausführen.

Abbildung 4.41
Abfrage ohne Funktionszeile und somit ohne Gruppierung

Die Einstellung in Abbildung 4.41 erzeugt die SELECT-Anweisung

```
SELECT tbKursbesuche.KID, tbKursbesuche.Zahlweise
FROM tbKursbesuche;
```

und liefert alle 18 Datensätze der Tabelle tbKursbesuche. Wird die Funktionszeile dagegen eingeblendet, kommen Sie zu der in Abbildung 4.42 gezeigten Einstellung, die zu einer vollständigen Gruppierung aller beteiligten Felder führt.

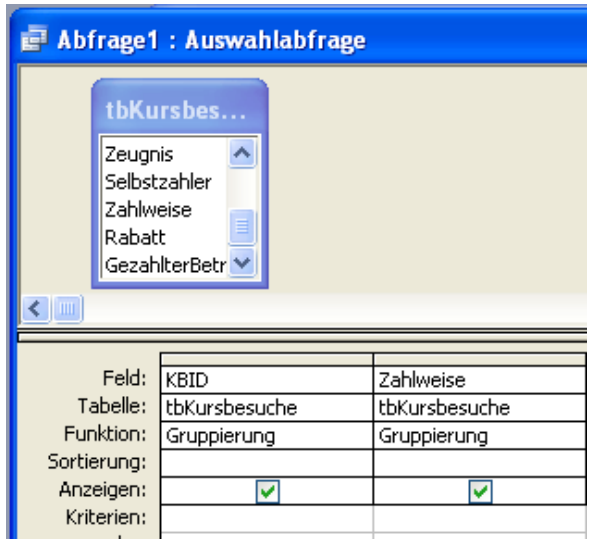


Abbildung 4.42
Mit eingeschalteter Funktion-Zeile erfolgt eine vollständige Gruppierung.

MS Access erzeugt daraus die SELECT-Anweisung:

```
SELECT tbKursbesuche.KID, tbKursbesuche.Zahlweise
FROM tbKursbesuche
GROUP BY tbKursbesuche.KID, tbKursbesuche.Zahlweise;
```

Listing 4.47
Totale Gruppierung in MS Access

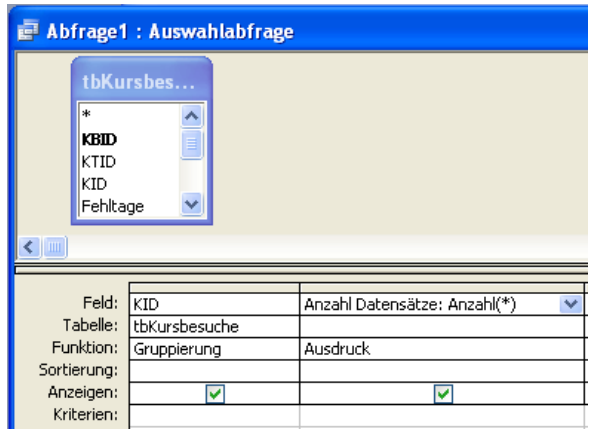
Diese SQL-Anweisung erzeugt folgerichtig zehn Datensätze, die den besprochenen zehn Gruppen entsprechen, wie sie in Abbildung 4.38 und Abbildung 4.39 dargestellt sind.

Noch ein Hinweis zu der speziellen COUNT(*)-Funktion, wie sie im Folgenden abgebildet ist.

```
SELECT kb.KID, Count(*) AS [Anzahl Kursteilnehmer]
FROM tbKursbesuche AS kb
GROUP BY kb.KID;
```

Diese Funktion lässt sich nicht in der gewohnten Weise eingeben. Hier muss auf die direkte Eingabe der Funktion in der obersten Zeile FELD zurückgegriffen werden. Die Syntax zeigt Abbildung 4.43. In gleicher Weise lassen sich alle Aggregatfunktionen verwenden. Beachten Sie die Einstellung *Ausdruck* in der Zeile FUNKTION, wenn Sie Aggregatfunktionen in dieser Form direkt eingeben. Wird die Aggregatfunktion bereits oben eingegeben, darf nicht noch einmal in der Funktionszeile aggregiert werden. Dies würde zu einer doppelten Aggregation führen, was verboten ist.

Abbildung 4.43
Aggregatfunktion direkt in der Feldbeschreibung



Der Einfluss der WHERE-Klausel

Beispiel mit WHERE

Die `WHERE`-Klausel wird vor der `GROUP BY`-Klausel abgearbeitet, die Datensätze werden also eingeschränkt, bevor sie gruppiert werden. Damit können Sie beispielsweise bestimmen, welche Gruppen überhaupt betrachtet werden, indem Sie nur auf die gewünschten Feldwerte des Gruppierungsfeldes mit der `WHERE`-Klausel filtern. Wir wollen das in einem weiteren Beispiel betrachten. Dabei wollen wir wissen, wie viele Personen in unserer Tabelle `tbPerson` in welcher Stadt wohnen, also nach dem Ort gruppieren. Dies soll aber nur für die Orte „Hannover“, „Braunschweig“ und „Celle“ geschehen. Zusätzlich wollen wir die Orte nach der Anzahl der Personen absteigend sortieren. Dazu können wir die folgende `SELECT`-Anweisung verwenden:

Listing 4.48
Gruppierung mit Filter

```
SELECT p.Ort, COUNT(*) AS "Personen in der Datenbank"
FROM tbPerson AS p
WHERE (p.Ort IN ('Hannover','Braunschweig','Celle'))
GROUP BY p.Ort
ORDER BY COUNT(*) DESC;
```

Sie sehen im Ergebnis in Abbildung 4.44, dass zunächst alle Datensätze bis auf die Einträge mit den drei gewünschten Städten herausgefiltert wurden. Dann sind die Datensätze gruppiert und entsprechend der Häufigkeit absteigend sortiert worden.

Abbildung 4.44
Ergebnis nach dem WHERE, GROUP BY und ORDER BY

Ort	Personen in der Datenbank
Celle	6
Hannover	5
Braunschweig	3

Info

Diejenigen, die schon eine Weile gerade auch mit klassischen Programmiersprachen wie COBOL oder PL/I programmiert haben oder die schon mit Reportgeneratoren gearbeitet haben, werden hier stark an den sogenannten „Gruppenwechsel“ erinnert worden sein, bei dem die Datensätze ebenfalls nach Gruppen sortiert und dann innerhalb der Gruppen einzeln verarbeitet werden. Tatsächlich bereitet SQL die Daten mithilfe der `GROUP BY`-Klausel ganz analog vor. Allerdings werden die einzelnen Sätze hier bereits zu einem kompletten Gruppendatensatz zusammengefasst. In einem weiterverarbeitenden Programm können die einzelnen Datensätze daher nicht mehr bearbeitet werden.

Zusammenfassung

Mit der `GROUP BY`-Klausel können Sie eine mehrstufige Gruppierung der Daten einer `SELECT`-Anweisung vornehmen. Die Gruppierung erfolgt nach der Filterung mit der `WHERE`-Klausel. Die Gruppierung kann alle Felder der Tabellen aus der `FROM`-Klausel umfassen. Die Gruppierungsfelder werden von links nach rechts genutzt, um die Ergebnisse in Gruppen und Teilgruppen zu zerlegen. Mit einem `COUNT(*)` können Sie die Anzahl der Datensätze pro Gruppe ermitteln. Mit anderen Gruppierungsfunktionen wie `SUM` (Summe) oder `AVG` (Mittelwert) können Sie Berechnungen bei der Aggregation von Datenfeldern durchführen.

Wichtige Aggregatfunktionen können Sie der folgenden Tabelle entnehmen. Beachten Sie, dass es Abweichungen zwischen den verschiedenen Datenbankmanagementsystemen und teilweise auch zwischen englischen und deutschen Versionen gibt. Genaueres finden Sie in Kapitel 5.

Tabelle 4.3 Wichtige Aggregatfunktionen

Aggregatfunktion	Bedeutung	grafische Oberfläche (deutsch)
`COUNT()`	Anzahl der Datensätze mit verschiedenen Werten eines Feldes, oder Anzahl der Datensätze `COUNT(*)`	Anzahl
`SUM()`	Summe der Datenfeldwerte	Summe
`AVG()`	Arithmetisches Mittel der Datenfeldwerte (Durchschnittswert)	Mittelwert
`MAX()`	Größter auftretender Wert	Max
`MIN()`	Kleinster auftretender Wert	Min
`STDDEV()`	Standardabweichung der Werte	StAbw
`VAR()`	Varianz der Werte	Varianz
`FIRST()`	Erster gefundener Wert (Wert des Datenfeldes im ersten Datensatz der Gruppe)	Erster Wert
`LAST()`	Letzter gefundener Wert (Wert des Datenfeldes im letzten Datensatz der Gruppe)	Letzter Wert

Damit beherrschen Sie jetzt SELECT-Anweisungen mit dieser Syntax:

SELECT-Syntax

```
SELECT [DISTINCT|ALL] feldname [AS alias] {, feldname [AS alias]}
FROM tabellen {joinliste}
[WHERE bedingungsliste]
[GROUP BY feldname [{,feldname}]]
[ORDER BY feldname [ASC|DESC]{ ,feldname [ASC|DESC]};
```

Übungen

Übungen zur SELECT-Anweisung mit GROUP BY-Klausel

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Ermitteln Sie aus allen Datensätzen in tbKursbesuche, wie viele Kursteilnehmer ohne Berücksichtigung des Kurses per Gutschein, bar beziehungsweise mit Überweisung bezahlen. Geben Sie die Zahlweise als eigenes Feld ebenfalls an. (Ü4.6.1)
2. Ermitteln Sie für alle Kursbesucher gruppiert nach Kursen, wie viele Selbstzahler dabei sind. Die anderen Datensätze sollen nicht interessieren. (Ü4.6.2)
3. Ermitteln Sie die Liste aus Ü4.6.1, aber sortieren Sie sie absteigend nach der Anzahl der Teilnehmer und beziehen Sie sich nur auf die Kurse „CE17“ und „CE23“. (Ü4.6.3)
4. Ermitteln Sie die Anzahl der Datensätze in der Tabelle tbKursbesuche insgesamt. (Ü4.6.4)
5. Ermitteln Sie je Kurs die Summe der Fehltage. Geben Sie die KID, das Kursthema und die Anzahl der Fehltage aus. (Ü4.6.5)

4.7 Die HAVING-Klausel

Stellen Sie sich vor, Sie wollen die Anzahl der Personen in Ihrer Datenbank ermitteln, die in einer bestimmten Stadt wohnen. Dann können Sie das mit einer GROUP BY-Klausel erreichen. Stellen Sie sich weiter vor, Sie wollen nur die Städte in Ihrer Ergebnismenge haben, in denen mindestens drei Personen wohnen. Dann müssen Sie alle Datensätze herausfiltern, die zu Orten mit weniger als drei Personen gehören. Das kann aber nicht mit einer WHERE-Klausel funktionieren, da diese vor der GROUP BY-Klausel abgearbeitet wird und der SQL-Interpreter daher zum Zeitpunkt der Bearbeitung der WHERE-Klausel noch gar nicht weiß, wie viele Datensätze zu einem Ort gehören werden.

Daher muss eine Möglichkeit geschaffen werden, die Ergebnismenge noch einmal zu filtern, nachdem die Gruppenbildung erfolgt ist. Dafür gibt es die HAVING-Klausel der SELECT-Anweisung.

Wir wollen jetzt also auch Eigenschaften von gruppierten Datensätzen nutzen, um die gewünschten Daten auszuwählen, also in Gruppendatensätzen filtern. Dazu nutzen wir die HAVING-Klausel der SELECT-Anweisung, um auf die gewünschten Datengruppen zu filtern.

Beispiel

Soll beispielsweise für jeden Ort in der Personentabelle tbPerson ermittelt werden, wie viele Personen dort wohnen, und sollen nur die Orte ausgegeben werden, in denen mindestens drei Personen wohnen, so kann dies mit der folgenden SELECT-Anweisung erreicht werden:

Listing 4.49
Orte mit mindestens drei Personen in der Datenbank

```
SELECT p.Ort, COUNT(*) AS "Personen in der Datenbank"
FROM tbPerson AS p
GROUP BY p.Ort
HAVING COUNT(*) >= 3
ORDER BY COUNT(*) DESC;
```

Das Ergebnis ist dasselbe wie das in der Abbildung 4.44. Es entsteht allerdings auf andere Art. War es oben eher Zufall, dass nur Gruppendatensätze mit mindestens drei Datensätzen berücksichtigt werden, wird dies jetzt gezielt konstruiert. Zunächst werden alle Datensätze der Tabelle berücksichtigt und es wird für jeden Ort eine Gruppe, ein Gruppendatensatz, gebildet. Dann wird für jeden Gruppendatensatz mit der HAVING-Klausel überprüft, ob er die Bedingung erfüllt, dass er aus mindestens drei Einzeldatensätzen gebildet wurde. Ist dies nicht der Fall, wird der entsprechende Gruppendatensatz aus der Ergebnisdatenmenge entfernt. Schließlich werden die verbleibenden Gruppen nach ihrer Größe absteigend sortiert.

HAVING-Syntax

HAVING (*bedingung*) { *AND* | *OR (bedingung)* }

Die Bedingung der HAVING-Klausel wird auf jede Gruppe der GROUP BY-Klausel angewendet, es wird also jede Gruppe überprüft. Stellen Sie sich das als Schleife über die einzelnen Gruppendatensätze vor. Nur die Gruppen, die die HAVING-Bedingung erfüllen, bleiben Teil der Ergebnismenge.

Die WHERE-Klausel und die HAVING-Klausel haben also viele Gemeinsamkeiten. Eigentlich verhalten sich beide Klauseln fast gleich. Der Unterschied besteht nur darin, dass sich die WHERE-Klausel auf einzelne Datensätze, die HAVING-Klausel auf Gruppendatensätze bezieht.

Wann sollte jetzt eine Filterbedingung wo platziert werden, WHERE oder HAVING? Eine Filterbedingung, die sich auf das Datenfeld der untersten Gruppierungsebene, also die Einzelsatzebene, bezieht, lässt sich immer in einer WHERE-Klausel unterbringen, wie bereits an obiger SELECT-Anweisung zu sehen war:

```
SELECT p.Ort, COUNT(*) AS "Personen in der Datenbank"
FROM tbPerson AS p
WHERE (p.Ort IN ('Hannover','Braunschweig','Celle'))
GROUP BY p.Ort
ORDER BY COUNT(*) DESC;
```

Derselbe Effekt lässt sich in diesem Fall auch mit einer HAVING-Klausel erreichen, wenn eine GROUP BY-Klausel existiert.

Listing 4.50
Wahlweises Filtern mit WHERE und HAVING

```
SELECT p.Ort, COUNT(*) AS "Personen in der Datenbank"
FROM tbPerson AS p
GROUP BY p.Ort
HAVING (p.Ort IN ('Hannover','Braunschweig','Celle'))
ORDER BY COUNT(*) DESC;
```

WHERE versus HAVING

In jedem anderen Fall, also wenn sich die Filterbedingung auf ein Feld des Gruppendatensatzes mit dem Zähler (COUNT) oder einer anderen Aggregatfunktion bezieht, ist die Nutzung der HAVING-Klausel statt der WHERE-Klausel zwingend erforderlich, weil der zu filternde Wert erst durch die Aggregation entsteht. Haben Sie wie im ersten Beispiel die Wahl, ist also eine HAVING-Klausel nicht zwingend, so sollten Sie im Normalfall die WHERE-Bedingung verwenden, um die Datenmenge so früh wie möglich zu reduzieren.

Beispiel

Ein weiteres Beispiel, in dem sowohl die WHERE- als auch die HAVING-Klausel verwendet werden können, ist eine Fortführung des Beispiels der GROUP BY-Klausel aus Listing 4.43. Hierbei wird die Anzahl der Teilnehmer an den Kursen ermittelt. Jetzt wird die Betrachtung aber auf den Kurs „CE23“ eingeschränkt. Hier gibt es zwei Varianten.

- Variante 1

```
SELECT kb.KID, COUNT(*) AS "Anzahl Kursteilnehmer"
FROM tbKursbesuche AS kb
GROUP BY kb.KID
HAVING (kb.KID='CE23');
```

- Variante 2

Listing 4.51
Varianten mit WHERE und HAVING

```
SELECT kb.KID, COUNT(*) AS "Anzahl Kursteilnehmer"
FROM tbKursbesuche AS kb
WHERE (kb.KID='CE23')
GROUP BY kb.KID;
```

Das Ergebnis beider Abfragen ist in Abbildung 4.45 zu sehen. Normalerweise ist die Variante 2 vorzuziehen.

Abbildung 4.45
Ergebnis der SELECT-Anweisung mit HAVING-Klausel

KID	Anzahl Kursteilnehmer
CE23	8

Tipp

Nutzen Sie bei größeren Datenbeständen, wann immer möglich, die WHERE-Klausel statt der HAVING-Klausel. Die WHERE-Klausel reduziert die Datenmenge frühzeitig. Damit sind bei der Gruppierung weniger Sortieroperationen notwendig und die Performance verbessert sich im Allgemeinen deutlich.

Zusammenfassung

Die HAVING-Klausel erlaubt die Formulierung von Filterbedingungen für die durch die GROUP BY-Klausel definierten Gruppendatensätze. Für jede Gruppe existiert ein Gruppendatensatz, dessen mit Aggregatfunktionen berechnete Felder die Anforderungen der HAVING-Klausel erfüllen müssen, damit der Gruppendatensatz Bestandteil der Ergebnismenge wird. Fehlt die HAVING-Klausel, werden alle Gruppendatensätze ermittelt.

Damit haben wir jetzt die zunächst vollständige Syntax der SELECT-Anweisung erarbeitet.

Syntax

SELECT [DISTINCT|ALL] *feldname [AS ALIAS]* { , *feldname [AS ALIAS]*}

FROM *tabelle* {*joinliste*}

[WHERE (*bedingung)* { *AND* | *OR (bedingung)* }*]*

[GROUP BY *feldname [*{, *feldname*}*]]*

[HAVING (*bedingung)* { *AND* | *OR (bedingung)* }*]*

[ORDER BY *feldname [ASC|DESC]* { ,*feldname [ASC|DESC]*}*];*

Diese Syntax entspricht dem Standardumfang aller relationalen Datenbankmanagementsysteme. Kleinere Abweichungen bei einzelnen der hier verwendeten Beispielsysteme haben wir bereits gesehen. Über den bis hierher definierten Umfang hinaus gibt es eine Reihe von Erweiterungen, die mehr oder weniger weitverbreitet sind und oft eigenen syntaktischen Regeln folgen. Die meisten Erweiterungen beziehen sich auf physische Eigenschaften der Datenbank und dienen der Steuerung der Datenabfrage sowie der Performance. Auf einige Punkte kommen wir an einigen Stellen noch zu sprechen, ohne auf den gesamten Umfang der einzelnen Systeme eingehen zu können.

Zwei wesentliche standardisierte Erweiterungen fehlen aber noch. Zum einen muss auf die Verwendung von Funktionen zur Berechnung oder Erstellung komplexerer Ausdrücke eingegangen werden, zum anderen fehlen die Unterabfragen. Die Funktionen sollen in folgenden Kapitel 5 im Zusammenhang mit den Datentypen behandelt werden. Unterabfragen, bei denen SELECT-Anweisungen geschachtelt oder als Teil anderer SQL-Anweisungen verwendet werden, werden ab Kapitel 9 behandelt.

Übungen zur SELECT-Anweisung mit HAVING-Klausel

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Ermitteln Sie aus der Tabelle `tbKursbesuche` je Kurs den kleinsten und größten Rabatt, der gewährt wurde. Geben Sie nur die Kurse an, bei denen der größte Rabatt über 50 liegt. Sortieren Sie das Ergebnis absteigend nach dem größten Rabatt. (Ü4.7.1)
2. Ermitteln Sie die KID und den durchschnittlichen Rabatt für alle Kursteilnehmer gruppiert nach Kursen. Geben Sie dabei nur diejenigen an, bei denen der durchschnittliche gezahlte Betrag der Teilnehmer kleiner als 250 ist. (Ü4.7.2)
3. Gruppieren Sie alle Kurse nach der Gebühr. Bestimmen Sie die Anzahl der Datensätze und geben Sie nur solche Gruppen aus, die aus mindestens zwei Datensätzen bestehen. Sortieren Sie das Ergebnis aufsteigend nach der Gebühr. (Ü4.7.3)
4. Ermitteln Sie die Anzahl der Kurse, die mindestens 300 € kosten und gruppieren Sie sie nach der Gebühr. (Ü4.7.4)

5

Datentypen, Ausdrücke und Funktionen

Sie haben jetzt bereits recht komplexe SELECT-Anweisungen erstellt. Dabei haben Sie aber immer die Felder so verwendet, wie diese in den Tabellen definiert sind. Sie ahnen es nach diesen Sätzen schon, es gibt auch noch weitere Möglichkeiten. Tatsächlich können neben den Feldern auch Konstanten (Literale) oder mithilfe der Felder berechnete Ausdrücke in SELECT-Anweisungen verwendet werden. Bevor wir dies hier ausführlich betrachten, müssen wir aber über Datentypen sprechen. War es der SELECT-Anweisung noch relativ egal, welche Daten dargestellt werden, so müssen wie in jeder Programmiersprache auch in SQL, spätestens, wenn es um die Verwendung von Funktionen geht, diese Datentypen berücksichtigt werden.

5.1 Datentypen

Beispiel

Um die Bedeutung von Datentypen zu zeigen, soll zunächst ein Beispiel genutzt werden, das bereits eine ganze Fülle von Funktionen verwendet, die von dem jeweiligen Datentyp der Felder abhängen.

Listing 5.1
Beispiel für ein komplexes SELECT mit Funktionen

```
SELECT
   CONCAT(p.Familienname, IF(p.Vorname IS NOT NULL,
   CONCAT(', ',p.Vorname),'')) AS "Name",
   EXTRACT(YEAR FROM p.Geburtsdatum) AS "Geburtsjahr",
   kb.KID,
   CONCAT(ROUND(kb.GezahlterBetrag/k.Gebuehr*100,2),' %')
   AS "Bezahlter Anteil"
FROM tbPerson p
   INNER JOIN tbKursbesuche kb ON (p.PID = kb.KTID)
   INNER JOIN tbKurs k ON (kb.KID = k.KID)
ORDER BY 1 ASC;
```

Zunächst einmal soll das Ergebnis der SELECT-Anweisung aus Listing 5.1 betrachtet werden. Dies ist in Abbildung 5.1 dargestellt. Sie sehen, dass zunächst der Familienname und der Vorname aneinandergehängt werden. Dies geschieht mit der Funktion CONCAT, die alphanumerische Daten erfor-

dert. Zusätzlich ist hier noch eine IF-Funktion eingebaut, die auf die Existenz von Werten prüft. Dafür sind sogenannte NULL-Werte wichtig, die in Abschnitt 5.2 besprochen werden.

Dann wird das Geburtsjahr als Jahr aus dem Geburtsdatum extrahiert. Dies funktioniert nur bei Feldern des Typs „Datum“, nicht bei numerischen oder alphanumerischen Feldern.

Anschließend wird der „bezahlte Anteil“ an den Kursgebühren berechnet. Derartige Berechnungen können natürlich nur mit numerischen Angaben erfolgen. Das Ergebnis wird anschließend in einem CONCAT mit dem Prozentzeichen kombiniert. Dafür wandelt es der SQL-Interpreter intern in eine alphanumerische Angabe um.

Abbildung 5.1
Ergebnis der SELECT-Anweisung in Listing 5.1

Name	Geburtsjahr	KID	Bezahlter Anteil
Karmann, Thomas	1954	CE17	100.00 %
Karmann, Thomas	1954	CE23	100.00 %
Karmann, Thomas	1954	CE24	62.50 %
Klötzer, Karl	1971	CE17	100.00 %
Klötzer, Karl	1971	CE23	14.29 %
Lisewski, Bernd	1960	CE24	87.50 %
Magerkurth, Melissa	1951	CE23	14.29 %
Martens, Melanie	1911	CE24	0.00 %
Martens, Melanie	1911	CE23	71.43 %
Müller, Claudia	NULL	CE24	0.00 %
Müller, Claudia	NULL	CE23	100.00 %
Peredy, Helmut	1956	CE24	75.00 %
Plate, Ulrich	1986	H90	66.67 %
Ruppert, Nicola	1962	CE23	100.00 %
Sander	1953	CE17	92.86 %
Schmidt, Karl	1949	H90	100.00 %
Schulze, Tanja	1992	CE23	100.00 %
Winter, Petra	1989	CE23	85.71 %

SQL-Anweisungen erlauben also eine große Vielfalt von Berechnungen und sonstigen Umformungen in den Ergebnismengen. Wir wollen uns nun mit den Datentypen beschäftigen, die SQL kennt und die bei diesen Umformungen zu beachten sind.

Grundsätzlich unterscheidet man in SQL:

- Alphanumerische Daten (zeichenorientiert)
- Numerische Daten (zahlenorientiert)
- Zeitbezogene Daten (Datum, Uhrzeit, Zeitintervalle)
- Binäre Daten (Daten mit eigener innerer Struktur, die nicht von SQL interpretiert werden)

In Tabelle 5.1 sind die Basisdatentypen zusammengestellt, die heute die Grundlage der gängigen Datenbanken bilden und weitgehend durch den SQL-Standard in seinen Entwicklungsstufen definiert sind.

Tabelle 5.1 Basisdatentypen der meisten SQL-Datenbanken

Kategorie	Datentyp	Beschreibung
alphanumerisch	`CHARACTER(n)`	Jede Art alphanumerischer Zeichen entsprechend dem eingestellten Zeichensatz. Es wird Speicherplatz für Zeichenketten der festen Länge `n` reserviert.
	`CHARACTER VARYING(n)`	Wie `CHARACTER`, aber es werden nur die tatsächlich verwendeten Zeichen belegt. `n` gibt die maximale Zeichenanzahl an.
numerisch	`NUMERIC(m,n)`	Dezimalzahl mit der festen Länge `m` und mit `n` Nachkommastellen, beispielsweise bei finanzmathematischen Berechnungen.
	`DECIMAL(m,n)`	Dezimalzahl mit variabler (maximaler) Länge `m` und mit `n` Nachkommastellen, ansonsten wie `NUMERIC`.
	`INTEGER`	Ganze Zahl (4 Byte)
	`SMALLINT`	Ganze Zahl (2 Byte)
	`BIGINT`	Ganze Zahl (8 Byte) ab SQL2003
	`FLOAT(n)`	Gleitkommazahl mit gegebener Genauigkeit `n`
	`REAL`	Gleitkommazahl mit einfacher Genauigkeit
	`DOUBLE [PRECISION]`	Gleitkommazahl mit doppelter Genauigkeit
Datum/Zeit	`DATE`	Datumsangabe, normalerweise `'JJJJ-MM-TT'`
	`TIME`	Zeitangabe, normalerweise `'hh:mm:ss'` (plus Zeitzone und weitere Dezimalstellen nach Implementierung)
	`TIMESTAMP`	Kombination von `DATE` und `TIME`
Intervall	`INTERVAL DAY`	Intervall Tage
	`INTERVAL HOUR`	Intervall Stunden

Tabelle 5.1 (Forts.) Basisdatentypen der meisten SQL-Datenbanken

Kategorie	Datentyp	Beschreibung
sonstige	BLOB	Binary Large Object. Speicherung binärer Daten beispielsweise für Grafiken. Der Zugriff erfolgt über spezielle Methoden, ab SQL99.
	CLOB	Character Large Object. Vergleichbar mit dem BLOB, aber für textbasierte Objekte, ab SQL99.
	BIT(n)	Bitkette der Länge n (normalerweise maximale Länge)
	BOOLEAN	Wahrheitswert Ja/Nein, True/False

Zu diesen Basistypen kommen in der Regel noch eine Reihe weiterer Datentypen, die datenbankspezifisch sind. Hierzu gehören streng genommen bereits die logischen Datentypen (Boolean), aber beispielsweise auch Aufzählungstypen, Mengentypen oder Konstrukte wie das neue MULTISET.

Darüber hinaus bieten viele Datenbanken strukturierte Datentypen, Felder (ARRAY) und diverse Typen für spezielle Anwendungszwecke wie beispielsweise Geodaten oder multidimensionale Strukturen.

CREATE TYPE

In einigen Systemen können mit einer eigenen SQL-Anweisung, CREATE TYPE oder CREATE DOMAIN, weitere Datentypen definiert werden. Die so definierten Datentypen können dann wie normale Datentypen bei der Definition von Tabellen eingesetzt werden. Beispielsweise kann ein Punkt im zweidimensionalen Raum mit

```
CREATE TYPE Punkt (x FLOAT, y FLOAT)
```

definiert und dann in Tabellen als Datentyp für geometrische Informationen genutzt werden.

Info

Die sogenannten BIT-Typen für binäre Daten in sind mit Erscheinen des SQL-Standards in der Version SQL2003 nicht mehr Teil des Standards, aber natürlich noch in vielen Datenbanksystemen implementiert.

Datentypcode

Die Vielfalt der Erweiterungen und unterschiedlichen Implementierungen der Datentypen, die die heutigen Datenbanksysteme bieten, stellen die Anwender vor einige Probleme. Diese Probleme sind relativ gering, solange Sie sich innerhalb eines Systems mit SQL bewegen, da der SQL-Interpreter auf die Datentypen abgestimmt ist. Soll aber über eine Programmierschnittstelle auf eine oder gar mehrere unterschiedliche Systeme zugegriffen werden, müssen die entsprechenden Beschreibungen der technischen Dokumentation genau beachtet werden. Ursprünglich sollten die Datentypen durch einen Datentypcode standardisiert werden. Der ANSI-Standard ordnet den Datentypen sogenannte Datentypcodes (DATATYPE CODE) zu. Beginnend mit CHARACTER=1, NUMERIC=2, DECIMAL=3 wird dabei den einzelnen Datentypen ein eindeutiger Code zugeordnet. Später kamen auch Untertypen hinzu,

sodass die Codes allein nicht mehr eindeutig sind. So hat beispielsweise bei den Zeitintervallen `INTERVAL DAY` den Code 10 und den Untertyp 3. `INTERVAL HOUR` hat ebenfalls den Typ 10, aber den Untertyp 4. Diese Datentypen können in einigen Systemen, insbesondere bei der Programmierung mit Schnittstellen, aus verschiedenen Programmiersprachen verwendet werden, bieten aber auch keine hundertprozentige Garantie für Kompatibilität.

Die Vielfalt der Datentypen erklärt sich aus dem Bestreben der Datenbankhersteller, den Anwendern Wahlmöglichkeiten bei

- der Größe der Datenfelder (Speicherplatz) und
- der Verarbeitung der Datenfelder (Funktionen)

zu bieten und so der Datenbank Vorteile zu verschaffen.

Speicherplatz

Natürlich ist Speicherplatz seit der Entwicklung der ersten SQL-fähigen Datenbanken erheblich günstiger und in größerem Umfang verfügbar geworden. Gleichzeitig sind aber auch die Anforderungen durch wachsende Datenmengen gestiegen. Spielt es bei unseren Beispieldatenbanken noch keine große Rolle, ob ein Namensfeld 25 oder 100 Zeichen lang ist, kann dies bei 10 Millionen Datensätzen allein in diesem Feld einen Unterschied von ungefähr 750 MB ausmachen. Multipliziert man dies mit der Anzahl der Datenfelder und Tabellen, mit Versionen und Sicherungsständen, macht es auch heute unter Umständen noch Sinn, bei der Wahl des Datentyps gezielt auch auf den Speicherbedarf zu achten.

Funktionen

Der zweite Aspekt liegt in den Funktionen, die für die einzelnen Datentypen zur Verfügung stehen. Soll beispielsweise mit den Inhalten eines Feldes gerechnet werden, stellen die numerischen Datentypen eine sinnvolle Option dar, kommt es mehr auf eine einfache und schnelle Formatierung an, kann ein alphanumerischer Datentyp sinnvoll sein.

5.1.1 Alphanumerische Angaben (Text)

Die auf den ersten Blick einfachsten Datentypen sind die alphanumerischen Angaben, also die Texte. Im einfachsten Fall werden dabei die Codes der einzelnen Zeichen eines Textes gespeichert. Wie ein Datenbanksystem dabei vorgeht, lässt sich an Listing 5.2 erkennen. Hier wird mithilfe der Funktion `ASC` der Code der ersten fünf Zeichen gemäß dem verwendeten Zeichensatz ermittelt und als Zahl dargestellt. Das Prinzip besteht also darin, einen Zeichensatz festzulegen und die einzelnen Zeichen dann jeweils numerisch als 2 Byte oder 4 Byte (Unicode) zu codieren.

Listing 5.2
Auswahl der ersten fünf Zeichen und deren ASCII-Codes

```
SELECT Familienname,
   SUBSTR(Familienname,1,1) AS Z1, ASCII(SUBSTR(Familienname,1,1))
   AS ASCII1,
   SUBSTR(Familienname,2,1) AS Z2, ASCII(SUBSTR(Familienname,2,1))
   AS ASCII2,
   SUBSTR(Familienname,3,1) AS Z3, ASCII(SUBSTR(Familienname,3,1))
   AS ASCII3,
   SUBSTR(Familienname,4,1) AS Z4, ASCII(SUBSTR(Familienname,4,1))
   AS ASCII4,
   SUBSTR(Familienname,5,1) AS Z5, ASCII(SUBSTR(Familienname,5,1))
   AS ASCII5
FROM tbPerson t;
```

Abbildung 5.2
ASCII-Codes der einzelnen Zeichen

Familienname	Z1	ASCII1	Z2	ASCII2	Z3	ASCII3	Z4	ASCII4	Z5	ASCII5
Weiss	W	87	e	101	i	105	s	115	s	115
Bucz	B	66	u	117	c	99	z	122		0
Karmann	K	75	a	97	r	114	m	109	a	97
Klötzer	K	75	l	108	ö	246	t	116	z	122
Weiss	W	87	e	101	i	105	s	115	s	115
Weiss	W	87	e	101	i	105	s	115	s	115
Meier	M	77	e	101	i	105	e	101	r	114
Schmidt	S	83	c	99	h	104	m	109	i	105
Müller	M	77	ü	252	l	108	l	108	e	101

Probleme

Problematisch sind Texte in der Praxis durch ihren Speicherbedarf, insbesondere bei sehr unterschiedlich langen Texten, durch den Zeichensatz, der festgelegt werden muss, durch die Unterscheidung in Groß-/Kleinschreibung und durch die oft schlechte Performance bei Textrecherchen.

CHAR vs. VARCHAR

Beginnen wir mit dem Speicherbedarf. In einer Datenbanktabelle hat ein bestimmtes Feld für jeden Datensatz eine festgelegte Länge. Soll in einem Feld ein Text gespeichert werden, muss der längste in allen Datensätzen zu erwartende Text auch noch gespeichert werden können. Entsprechend muss das Textfeld für diesen größten Wert dimensioniert werden. Dies kann zu sehr großen Feldern führen, die noch dazu in vielen Datensätzen nur sehr wenige Zeichen beinhalten, nur um im ungünstigsten Fall noch ausreichend Speicherplatz zu bieten. Daher hat man bereits frühzeitig im Standard eine Unterscheidung zwischen Textfeldern mit fester Länge (CHAR) und solchen mit variabler Länge (VARCHAR) getroffen. Bei CHAR-Feldern wird stets die volle Anzahl der Bytes belegt, die bei der Definition angegeben ist, ob sie nun benötigt wird oder nicht. Nicht benötigte Zeichen werden zumeist mit dem Code „0“ belegt (siehe Abbildung 5.2 zweiter Datensatz). Bei VARCHAR-Feldern wird die tatsächlich benötigte Länge in einem oder mehreren zusätzlichen Bytes gespeichert und nur die tatsächlich benötigte Speicherplatzgröße belegt. CHAR-Felder sind auf eine maximale Zeichenanzahl begrenzt, die Anzahl liegt je nach Datenbank zumeist zwischen 255 und 65.535 Zeichen. VARCHAR erlaubt (auch wieder zumeist) mehr Zeichen, je nach Datenbank zwischen 65.535 Zeichen und 2 GB.

Nun böte es sich an, stets VARCHAR zu verwenden, da dies mit Ausnahme sehr kleiner Datenfelder doch wesentlich speicherplatzeffizienter ist. Der Nachteil liegt allerdings für die Datenbank in der unterschiedlichen Länge von Datensätzen, die so entstehen, und den zusätzlichen Zugriffen auf die Längenbytes. Daher werden diese Felder oft datenbankintern getrennt gespeichert und haben Einschränkungen und Performance-Nachteile bei der Suche mit SELECT-Anweisungen. Letztlich ist hier also immer zwischen Speicherplatz und Performance abzuwägen, was in der Informatik keine wirklich neue Erkenntnis ist. Im Zweifelsfall sollten Sie sich bei kleinen und mittleren Datenbanken für die feste Länge entscheiden.

TEXT, CLOB

Bieten VARCHAR-Felder bereits komfortablen Speicherplatz, so kann dieser in Fällen, in denen Sie komplette Texte in der Datenbank ablegen wollen, ebenfalls noch nicht ausreichend sein. Für besonders lange Texte sind daher Erweiterungen geschaffen worden, bei denen Texte in großen Containern abgelegt werden, die mit SQL zumeist nur noch begrenzt bearbeitbar sind. Diese Container tragen je nach Datenbank Namen wie „TEXT“, „Memo“ oder „CLOB“ (Character Large OBject). Im Endeffekt sind Felder dieser Datentypen letztlich nur in die Datenbank eingelagerte Container mit dem Vorteil, alle Informationen zentral in einem System verwalten zu können. Die Zugriffsmöglichkeiten mit SELECT-Anweisungen sind sehr eingeschränkt, insbesondere ist oft keine Suche mit einem

```
WHERE feldname LIKE '%suchtext%'
```

möglich. Einige neuere – wiederum datenbankspezifische – Entwicklungen bieten hier aber immer mehr Bearbeitungsmöglichkeiten inklusive einer Volltextsuche.

Zeichensatz

Neben der reinen Größe von Textobjekten entstehen auch durch den verwendeten Zeichensatz Probleme. In CHAR- und VARCHAR-Feldern werden Texte gemäß dem für die Datenbank gewählten oder bei der Definition der Tabelle angegebenen Zeichensatz gespeichert. Dies setzt dann voraus, dass die Datenbank stets mit demselben Zeichensatz verwendet wird oder dass eine entsprechende Umwandlung erfolgt. Der Vorteil ist in jedem Fall, dass bekannt ist, mit welchem Zeichensatz gearbeitet wird, und jedes mit diesem Zeichensatz darstellbare Zeichen auch gespeichert werden kann.

Spielt der Zeichensatz keine Rolle, können Texte auch platzsparender binär statt byteweise gespeichert werden. Dafür stehen Typen wie BINARY oder VARBINARY zur Verfügung. Diese lassen sich dann zumeist mit CONVERT in verschiedenen Zeichensätzen interpretieren. Den Unterschied können Sie sich mithilfe eines kleinen Beispiels klarmachen.

```
SELECT Familienname, SUBSTR(Familienname,3,1) AS Z3,
ASCII (SUBSTR(Familienname,3,1)) AS ASCII1,
CONVERT (SUBSTR(Familienname,3,1) USING utf8) AS ASCII2
FROM tbPerson t;
```

Listing 5.3
Bei zeichenbasierten „normalen“ Strings erfolgt keine Konvertierung (MySQL).

```
SELECT Familienname, SUBSTR(Familienname,3,1) AS Z3,
ASCII (BINARY (SUBSTR(Familienname,3,1))) AS ASCII1,
CONVERT (BINARY (SUBSTR(Familienname,3,1)) USING utf8) AS ASCII2
FROM tbPerson t;
```

Listing 5.4
Bei binären Strings wird in einen neuen Zeichencode umgesetzt (MySQL).

In beiden Listings wird versucht, den dritten Buchstaben des Familiennamens zu ermitteln. Dabei wird einmal das Zeichen ohne Konvertierung ermittelt, was problemlos in beiden Beispielen funktioniert, da der Zeichensatz verwendet wird, der auch bei der Eingabe eingesetzt wurde.

Dann wird einmal der Inhalt als ASCII-Zeichencode ausgegeben, der in beiden Beispielen ebenfalls dasselbe Ergebnis liefert. Schließlich wird das Zeichen konvertiert in den Zeichensatz UTF8 ausgegeben. Die Daten sind jeweils zeichenorientiert abgespeichert. Im zweiten Beispiel wird mit der

Funktion BINARY eine binäre Speicherung simuliert und das Ergebnis dann in UTF8 umgewandelt. Sie sehen, dass das „ö“ bei der binären Speicherung im Gegensatz zu der ersten zeichensatzbasierten Umwandlung jetzt nicht mehr umgesetzt werden kann.

Abbildung 5.3
Ergebnis ohne Zeichensatzumwandlung

Familienname	Z3	ASCII1	ASCII2
Weiss	i	105	i
Bucz	c	99	c
Karmann	r	114	r
Klötzer	ö	246	ö
Weiss	i	105	i

Abbildung 5.4
Ergebnis mit Zeichensatzumwandlung

Familienname	Z3	ASCII1	ASCII2
Weiss	i	105	i
Bucz	c	99	c
Karmann	r	114	r
Klötzer	ö	246	NULL
Weiss	i	105	i

Die Tabelle 5.2 bietet eine Übersicht über die Namen der beiden wichtigsten alphanumerischen Datentypen in den verschiedenen Systemen.

Tabelle 5.2
Alphanumerische Datentypen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
CHARACTER(n) max. n	CHAR(n) 255	CHAR(n) (Text) 255	CHAR(n) 4000	CHAR (n) 32.767	CHAR(n)
VARYING CHARACTER(n) max. n	VARCHAR (n) 65.532	VARCHAR(n) (Memo) 64.000	VARCHAR2 (n) 4.000	VARCHAR (n) 32.765	VARCHAR (n)

Viele Datenbanken bieten zusätzliche Typen, um die national relevanten Zeichensätze direkt zu verwenden. Diese heißen dann beispielsweise NCHAR oder NVARCHAR. Im Normalfall werden sie aber nur aus Kompatibilitätsgründen mit Oracle zugelassen und haben kein besonderes Verhalten, sodass auf sie oft verzichtet werden kann.

5.1.2 Ganze Zahlen

Eigentlich sollte man erwarten, dass für ganze Zahlen ein Datentyp ausreicht. Unterscheidet aber bereits der Standard verschiedene „INTEGER“, so können Sie am Beispiel MySQL, das eine große Vielfalt von ganzzahligen Datentypen bietet, sehen, wo der Unterschied im Einzelnen liegt. Bei den Datentypen für ganze Zahlen fallen insbesondere die zusätzlichen Typen TINYINT und MEDIUMINT auf (siehe Tabelle 5.3).

Tabelle 5.3
Übersicht über ganzzahlige Typen in MySQL

Datentyp	Byte	Unsigned	kleinster Wert	größter Wert
TINYINT	1	Nein	-128	127
		Ja	0	255
SMALLINT	2	Nein	-32.768	32.767
		Ja	0	65.535
MEDIUMINT	3	Nein	-8.388.608	8.388.607
		Ja	0	16.777.215
INT (INTEGER)	4	Nein	-2.147.483.648	2.147.483.647
		Ja	0	4.294.967.295
BIGINT	8	Nein	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
		Ja	0	18.446.744.073.709.551.615

Die wesentlichen Unterschiede liegen in der Größe der speicherbaren Zahlen und in dem dafür benötigten Speicherplatz in Byte. Bedenken Sie gerade bei Tabellen, die viele Datensätze beinhalten, werden, dass der Speicherbedarf sehr groß werden kann. Sie sehen, dass die Unterschiede erheblich sind.

Ganze Zahlen können in MySQL zusätzliche Angaben enthalten:

INTEGER [(*Stellenanzahl)] [UNSIGNED] [ZEROFILL]*

SIGNED | UNSIGNED

Die Unterscheidung zwischen `SIGNED` (Standard) für Zahlen mit Vorzeichen und `UNSIGNED` für Zahlen ohne Vorzeichen, die dann dank des eingesparten Bits für das Vorzeichen zumeist doppelt so große positive Zahlen erlauben, ist in IT-Sprachen weitverbreitet.

Die optionale Angabe der Stellenanzahl gibt die Anzahl dargestellter Ziffern an. Fehlen Ziffern, wird die Angabe links mit Leerzeichen aufgefüllt, was die Verwendung in Tabellen deutlich verbessert. Mit `ZEROFILL` kann auch mit führenden Nullen aufgefüllt werden. So wird bei der Angabe

```
INTEGER(5) ZEROFILL
```

die Postleitzahl 01438 korrekt mit führender Null angegeben. Die Tabelle 5.4 gibt eine Übersicht über die Standardtypen ganzer Zahlen in den verschiedenen Systemen.

Tabelle 5.4
Ganze Zahlen

ANSI	MySQL	MS Access	Oracle	Firebird	open-Base
SMALLINT (2 Byte)	SMALLINT	INTEGER (Zahl)	SMALLINT	SMALLINT	SMALLINT
INTEGER (4 Byte)	INTEGER (INT)	LONG INTEGER (Zahl)	INTEGER	INTEGER	INTEGER
BIGINT (8 Byte)	BIGINT	%	BIGINT	%	BIGINT

Oracle

Oracle stellt eine Besonderheit dar, da dort für Zahlen generell der Typ NUMBER empfohlen wird. NUMBER ist eine Oracle-Entwicklung, die in diesem Umfeld aber den De-facto-Standard darstellt. Für ganze Zahlen empfiehlt sich hier generell die Verwendung des Typs NUMBER(38).

Access

Die MS Access-Oberfläche bietet für alle numerischen Angaben die Auswahl *Zahl* an. Wird im unteren Auswahlbereich unter FELDGRÖSSE *Byte* (1 Byte), *Integer* oder *Long Integer* gewählt, ergibt sich daraus eine entsprechende Ganzzahl

5.1.3 Gleitkommazahlen

Die Zahlen mit Nachkommastellen lassen sich grundsätzlich in die Gleitkommazahlen mit einer variablen, von der Byte-Anzahl abhängigen oder einer festgelegten Genauigkeit und in Dezimalzahlen (manchmal auch Festkommazahlen) mit einer festen Anzahl Nachkommastellen einteilen.

Die erstere Gruppe entspricht den aus anderen Programmiersprachen wie C++ oder Pascal bekannten Implementierungen von Gleitkommazahlen. Es wird im Wesentlichen zwischen der Rechnung mit einfacher Genauigkeit (FLOAT) und mit doppelter Genauigkeit (DOUBLE PRECISION) unterschieden. Eine Besonderheit ist die Möglichkeit, die Nachkommastellen vorzugeben (FLOAT(n)).

Gleitkommazahl

Eine Gleitkommazahl oder Festkommazahl wird in der Form

$$(-1)^{\text{Vorzeichen}} * \text{Mantisse} * \text{Basis}^{\text{Exponent}}$$

dargestellt. Der wesentliche Unterschied zwischen der Gleitkommazahl und der Festkommazahl ist nur der, dass bei einer Gleitkommazahl binär gerechnet wird, die Basis also 2 ist, während bei einer Festkommazahl (Dezimalzahl) zur Basis 10 gerechnet wird. In beiden Fällen könnte mit einer flexiblen oder mit einer festen Anzahl von Nachkommastellen gearbeitet werden. Die binäre Logik ist aber schneller, erlaubt bei komplexen Berechnungen eine höhere Anzahl von Nachkommastellen und wird zumeist bei mathematischen Berechnungen bevorzugt, wo man dann zumeist auch die größtmögliche Genauigkeit nutzt. Es gibt sie aber mit variabler und (seltener genutzt) fester Anzahl von Nachkommastellen.

Dezimalzahl

Die Dezimalzahlen werden im kaufmännischen Bereich bevorzugt, da sie eher eine Abbildung der kaufmännischen Rundungsvorschriften erlauben. Hier wird entsprechend auch mit einer festen Anzahl an Nachkommastellen gerechnet. Die interne Darstellung ist ähnlich, erfolgt aber zur Basis 10, also entsprechend unserem gewohnten Zahlensystem.

Die Zahl 7.32 kann dann beispielsweise mit Vorzeichen = 0 (1 Bit), Mantisse 732 zur Basis 10 mit dem Exponent -2 wie folgt dargestellt werden.

$(-1)^0 * 732 * 10^{-2} = 7.32$

Für den kaufmännischen Bereich, insbesondere für Geldbeträge, werden also die Dezimalzahlen angeboten, bei denen zwischen solchen mit fester Länge (`NUMERIC`) und solchen mit variabler Länge (`DECIMAL`) unterschieden wird. In beiden Fällen kann die Anzahl der Nachkommastellen fest vorgegeben werden.

In Tabelle 5.5 wird ein Überblick über die Namen der gängigen Datentypen für Gleitkommazahlen und Dezimalzahlen gegeben.

Tabelle 5.5 Gleitkommazahlen und Festkommazahlen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
`NUMERIC (m,n)`	`NUMERIC (m,n)`	`CURRENCY` (Währung)	`NUMERIC (m,n)`	`NUMERIC (m,n)`	`NUMERIC (m,n)`
`DECIMAL (m,n)`	`DECIMAL (m,n)`	`DECIMAL (m,n)` (Dezimal)	`DECIMAL (m,n)`	`DECIMAL (m,n)`	`DECIMAL (m,n)`
`REAL` 4 Byte 7 Nachkommast.	`REAL`	`SINGLE, REAL` (Zahl)	`FLOAT(63)` `BINARY_FLOAT`	`FLOAT`	`REAL`
`DOUBLE PRECISION` 8 Byte 14 Nachkommast.	`FLOAT` (zusätzlich `DOUBLE`)	`DOUBLE, FLOAT` (Zahl)	`FLOAT(126)` `BINARY_DOUBLE`	`DOUBLE PRECISION`	`DOUBLE`
`FLOAT(n)`	`FLOAT(n)`	`SINGLE/ DOUBLE` (Zahl)	`FLOAT (n)`	%	`FLOAT(n)`

MySQL

Bei den Gleitkommazahlen steht zwar `NUMERIC` zur Verfügung, ist aber wie `DECIMAL`, also mit variabler Länge, implementiert.

Oracle

Eine Besonderheit von Oracle ist der Typ `NUMBER(m, n)`, der in Oracle-Datenbanken der numerische Typ schlechthin ist und für fast alle Zwecke verwendet werden kann. Er ähnelt am meisten dem `DECIMAL(m, n)`, aber durch das Weglassen der Nachkommastellen kann er auch für andere Zahlen, beispielsweise `INTEGER`, verwendet werden.

MS Access

Bei Access erfolgt die Einstellung des Typs über die Auswahl *Zahl* und die anschließende Auswahl der FELDGRÖSSE, die dann wiederum je nach Einstellung noch eine Auswahl der GENAUIGKEIT (m) und der DEZIMALSTELLEN (n) erlaubt. Der Datentyp WÄHRUNG wird direkt ausgewählt und erlaubt nur eine Angabe der Nachkommastellen. Durch seine Bindung an das Währungssymbol entspricht er nicht wirklich dem Typ `NUMERIC`, zeigt aber ein entsprechendes Verhalten.

5.1.4 Datum/Uhrzeit

Es gibt eine Reihe von Formaten, die speziell für die Speicherung von Datums- und Zeitangaben vorgesehen sind. Natürlich könnten diese Angaben auch in normalen alphanumerischen Feldern gespeichert werden. Die speziellen Datentypen wie `DATE`, `TIME`, `DATETIME` oder `TIMESTAMP` bieten aber den großen Vorteil, dass für sie spezielle Sortierungen und Berechnungen mit Funktionen in Datumslogik nutzbar sind.

Datum

Damit dies möglich ist, müssen alle Datumsangaben immer in demselben Format gespeichert werden. Normalerweise ist dies die Reihenfolge „Jahr-Monat-Tag“ für Datumsangaben mit zwei- oder vorzugsweise vierstelligen Jahresangaben und Bindestrichen als Trennzeichen. Die Angabe für Weihnachten 2008 wäre dann „2008-12-24“. Zweistellige Jahresangaben sind ebenfalls erlaubt, also „08-12-24“, und werden gleichbedeutend interpretiert. Bei zweistelligen Jahresangaben ist allerdings der Jahrtausendwechsel zu beachten. So kann 70 als Jahresangabe sowohl 1970 als auch 2070 bedeuten. Ab welchem Wert eine zweistellige Jahresangabe als 19xx interpretiert wird und bis wohin sie als 20xx gilt, ist einstellungsabhängig. MySQL hat zur Zeit als Standardeinstellung festgelegt, dass der Zeitraum 1970–2069 als gültig angesehen wird, also bedeutet „70“ noch „1970“ während „69“ schon für „2069“ steht. Testen Sie dies für Ihre Datenbank, wenn Sie mit zweistelligen Jahresangaben arbeiten, sonst kann es bei der Sortierung zu überraschenden Ergebnissen kommen. Generell sind vierstellige Jahresangaben vorzuziehen.

Uhrzeit

Für die Uhrzeit gilt normalerweise das Format „HH:MM:SS“, also jeweils zweistellige Angaben für Stunden, Minuten und Sekunden. Die Stunden werden im 24-Stunden-System verwaltet, Bruchteile von Sekunden müssen gesondert eingestellt werden.

Für die Abspeicherung sieht der Standard ein Format `DATE` vor, das sowohl das Datum als auch die Uhrzeit beinhaltet, also eine Angabe in der Form:

```
'JJJJ-MM-TT HH:MM:SS'
```

DATE, TIME, DATETIME

Bei der Umsetzung gibt es im Wesentlichen zwei Varianten. Entweder wird das Format – wie beispielsweise in Oracle – gemäß dem Standard umgesetzt oder es werden für Datum und Uhrzeit getrennte Formate, `DATE` und `TIME`, geschaffen, die dann für die Speicherung des Datums beziehungsweise der Uhrzeit verwendet werden. Manche Datenbanken wie MySQL bieten dann zusätzlich noch einen Datentyp `DATETIME`, um beides zu speichern.

TIMESTAMP

Ein besonderer Datentyp ist der Zeitstempel (TIMESTAMP). Dabei handelt es sich prinzipiell ebenfalls um eine Kombination von Datum und Uhrzeit. Da der Hauptzweck des Zeitstempels zumeist die eindeutige Markierung von Einfüge- oder Änderungszeitpunkten ist, gibt es noch die Möglichkeit, die Zeit durch die Angabe von Sekundenbruchteilen und/oder der Zeitzone eindeutiger zu gestalten.

Einige Datenbanken erlauben bis zu neun Nachkommastellen bei der Sekundenangabe, so kann man bei Oracle mit

```
TIMSTAMP(6)
```

beispielsweise Zeitangaben wie

12:23:15.362562

zulassen, also eine auf die Millionstelsekunde genaue Angabe, wenn dies denn die Rechneruhr hergibt.

GMT

Die Zeit (TIME) wird wie gesagt in einem 24-Stunden-System mit jeweils 2 Stellen für die Stunden, Minuten und Sekunden angegeben. Um eine weltweite Verarbeitung zu erleichtern, kann zusätzlich die Zeitzone angegeben werden. Mit dieser Korrektur kann die lokale Zeitangabe auf die Greenwich Mean Time (GMT) umgerechnet werden. Damit können alle Angaben entsprechend der lokalen Zeitzone gemacht werden. Für die deutsche Zeit wäre also jeweils eine Angabe wie +01:00 oder +1:00 anzufügen.

UTC

Inzwischen ist es üblich geworden, statt von der früher vertrauten GMT von der UTC (Universal Time Coordinate) zu sprechen, was derselben Zeitzone entspricht.

Die Zeitangabe

```
'2008-05-17 14:02:04 +1:00'
```

wäre also eine Angabe nach deutscher Standardzeit und entspräche

```
'2008-05-17 13:02:04 UTC'.
```

Die Sommerzeit (Daylight Saving Time) wird über individuelle Einstellungen berücksichtigt.

WITH TIMEZONE

Es gibt zu dem Datentyp TIMESTAMP noch die Erweiterung TIMESTAMP WITH TIMEZONE, die die Zeitdifferenz zwischen der lokalen Zeit und der UTC berücksichtigt, sodass alle Angaben weltweit vergleichbar werden. In diesem Fall wird die Zeit wie oben beschrieben abgespeichert, letztlich in UTC. Damit sind weltweit Daten unmittelbar vergleichbar.

Oracle bietet darüber hinaus noch TIMESTAMP WITH LOCAL TIMEZONE , wobei ebenfalls der Unterschied zwischen der lokalen Zeit und der UTC berücksichtigt wird, aber alle Angaben bei einer Abfrage jeweils in die lokale Zeit umgerechnet angezeigt werden.

Der Zeitstempel (TIMESTAMP) ist als internes Systemfeld sehr beliebt, dessen Wert bei Neueingaben oder Änderungen automatisch in die Datensätze eingefügt wird, um die jeweils letzte Änderung dokumentieren zu können.

Intervall

Genutzt werden neben den Angaben von Zeitpunkten auch die Angaben von Zeitintervallen, einige Datenbanken bieten hierzu auch eigene Datentypen. Ein Intervall kann beispielsweise in Oracle in einem Datentyp `INTERVAL DAY TO SECOND` gespeichert werden, der einen Zeitunterschied, ausgedrückt in Tagen, Stunden, Minuten und Sekunden, speichert. Die Tabelle 5.6 gibt einen Überblick über die wichtigsten Datums-/Uhrzeittypen in den verschiedenen Systemen.

Tabelle 5.6 Datentypen für Datums- und Zeitangaben

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
`DATE`	`DATETIME` `DATE` und `TIME` existieren auch einzeln	`DATE/TIME/ DATETIME` (Datum/Uhrzeit)	`DATE`	`DATE` und `TIME` zusammen	`DATE` und `TIME` zusammen
`TIMESTAMP`	`TIMESTAMP`	`TIMESTAMP` (Datum/Uhrzeit)	`TIMESTAMP(n)`	`TIMESTAMP`	`TIMESTAMP (n)` n=0 oder 6
`TIMESTAMP WITH TIMEZONE`	%	%	`TIMESTAMP WITH TIMEZONE`	%	%

5.1.5 BITs, BLOBs und andere Datentypen

Neben den beschriebenen Datentypen gibt es eine Reihe von Erweiterungen, die in vielen Systemen verfügbar sind. Die wichtigsten sind die Möglichkeit, binäre Werte, also 0/1, True/False, zu speichern, wofür zumeist nur 1 Bit benötigt wird, sowie die Möglichkeit, große Binärdateien beispielsweise für Multimedia-Objekte wie Bilder, Filme oder Musik zu speichern.

Tabelle 5.7 Sonstige gängige Datentypen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
`BLOB`	`BLOB` 64 k `LONGBLOB` 2 GB	`OLE Object` 64k	`BLOB, CLOB` 4 GB	`BLOB` 64 k	`BINARY, VARBINARY, LONGVARBINARY`
`BIT`	`BIT(n)`	`BINARY` teilweise `BOOLEAN` (Ja/Nein)	%	%	`BOOLEAN` (Ja/Nein)

Bei binären Werten kann beispielsweise in MySQL auch eine größere Bit-Anzahl definiert werden, also beispielsweise `BIT(8)`. Generell wird aber in den neuen SQL-Standards von der Verwendung dieser binären Datentypen abgeraten.

Auf BLOB-Daten kann mit den Standard-SQL-Anweisungen nicht zugegriffen werden. Die BLOB-Typen wurden hauptsächlich eingeführt, um derartige Daten nicht außerhalb der Datenbanken speichern zu müssen. Neben den reinen BLOB-Daten gibt es inzwischen ganze Bereiche, wo die Datentypen um XML-Daten, geografische Daten oder multidimensionale Daten mit den zugehörigen Erweiterungen des Standard-SQL in die Datenbanksysteme integriert wurden. Diese Erweiterungen sind aber zum einen datenbankspezifisch und zum anderen so komplex, dass sie den hier besprochenen Umfang von SQL bei Weitem übersteigen.

5.2 NULL-Werte

Eine Besonderheit von SQL ist der Wert NULL. NULL wird immer groß geschrieben und bedeutet immer das Fehlen eines Wertes, also das Fehlen einer Information, was streng genommen, selbst wieder eine Information ist. NULL kann bei allen Datentypen vorkommen.

Beispiel

Sie wollen die Personentabelle tbPerson auf Vollständigkeit der Angaben überprüfen. Sicher haben Sie schon bemerkt, dass zum Familiennamen „Sander“ der Vorname fehlt. Vielleicht ist Ihnen in Abbildung 5.5 auch schon aufgefallen, dass anstelle des Vornamens nicht einfach ein leerer Eintrag zu sehen ist, sondern klein in der Ecke „NULL“ eingeblendet ist (MySQL).

PID	Familienname	Vorname	PLZ	Strasse	Ort	Geburtsdatum
1	Weiss	Peter	30529	Palmstraße 6	Hannover	1963-11-07
2	Bucz	Susanne	30531	Heinestraße 23	Hannover	1976-04-06
4	Karmann	Thomas	29227	Trift 28	Celle	1954-08-04
5	Klötzer	Karl	29221	Bahnhofstraße 2	Celle	1971-03-13
6	Weiss	Karin	30529	Palmstraße 6	Hannover	1962-10-05
7	Weiss	Peter	38134	Glanweg 4	Braunschweig	1974-03-02
8	Meier	Kathrin	38154	Welfenallee 23	Braunschweig	1981-05-03
9	Schmidt	Karl	30529	Lavesallee 23	Hannover	1949-06-25
10	Müller	Claudia	29596	In den Fuhren 12	Breitenhees	NULL
11	Lisewski	Bernd	30890	Roggenkamp 10	Barsinghausen	1960-06-06
23	Peredy	Helmut	29221	Mauernstraße 2	Celle	1956-02-23
17	Schlachter	Dieter	29227	Mondhagen 43	Celle	1961-02-02
15	Martens	Melanie	29221	Horstweg 258	Celle	1911-02-17
24	Ruppert	Nicola	29301	Welfenallee 23	Bergen	1962-02-25
25	Sander	NULL	29223	Marxallee 12	Celle	1953-02-05
26	Cromberg	Jörg	38152	Nordring 13	Braunschweig	1991-06-07
31	Schulze	Tanja	29308	Berliner Ring 23	Winsen	1992-11-09
37	Magerkurth	Melissa	29336	Am Tümpel 3	Nienhagen	1951-09-04
32	Winter	Petra	29320	Immenhoop 51	Hermannsburg	1989-12-30
34	Plate	Ulrich	30529	Gutenberggasse 5	Hannover	1986-12-02

Abbildung 5.5
Tabelle mit NULL-Werten

Mit NULL wird zwischen einem leeren Wert, beispielsweise dem leeren Text '' und dem Fehlen eines Wertes unterschieden. NULL bedeutet bei einem Vornamen, dass der Name nicht bekannt ist, dass die Information fehlt. Ein leerer Text würde dagegen bedeuten, dass bekannt ist, dass kein Vorname existiert.

Listing 5.5
Abfrage auf Personen mit einem leeren Text als Vorname

```
SELECT p.PID, p.Familienname, p.Vorname
FROM tbPerson p
WHERE (p.Vorname='');
```

Das Ergebnis der SQL-Anweisung in Listing 5.5 ist eine leere Menge, es gibt keinen Datensatz, bei dem der Vorname ein leerer Text ist.

Der NULL-Wert des Vornamens des Datensatzes mit der PID „25“ entspricht also auch im Verständnis des SQL-Interpreters nicht dem leeren Text.

Listing 5.6
Abfrage auf alle Personen mit fehlendem Vornamen

```
SELECT p.PID, p.Familienname, p.Vorname
FROM tbPerson p
WHERE p.Vorname IS NULL;
```

Dagegen liefert die Anweisung in Listing 5.6 den erwarteten Datensatz, wie Sie ihn in Abbildung 5.6 sehen.

Abbildung 5.6
Ergebnis der SELECT-Anweisung mit Vorname IS NULL

pid	familienname	vorname
25	Sander	NULL

IS [NOT] NULL

NULL ist ein eigener Wert in SQL, er entspricht NICHT dem leeren Text '', nicht der 0 oder 0.0 oder einem Datum 0000-00-00. Die Abfrage auf NULL-Werte erfolgt mit dem eigenen Operator IS NULL beziehungsweise IS NOT NULL.

Durch wahlweise Nutzung und Abfrage von NULL-Werten können Sie zwischen dem Fehlen einer Information und der Information, dass etwas leer oder „0“ ist, unterscheiden. Haben Sie beispielsweise Dozenten, die nicht auf Stundenbasis, sondern mit einem Festgehalt bezahlt werden, kann es Sinn machen, dass Sie in diesen Fällen als Stundensatz 0.00 € eingeben. Sie dokumentieren damit, dass Sie wissen, dass kein Stundensatz existiert. Demgegenüber bedeutet dann die Eingabe „NULL“, dass zurzeit keine Informationen über den Stundensatz vorliegen.

5.3 Literale

Wir haben bisher schon oft Werte von Datenfeldern mit festen Werten verglichen. So haben wir Ausdrücke wie

```
k.KID='CE23'
```

oder

```
k.GezahlterBetrag < 250
```

formuliert, ohne dies weiter zu kommentieren.

Ausdrücke, die lediglich feste Werte und keine Datenbankfelder beinhalten, beispielsweise 'Celle', 25, 4+5 oder 12.4.2008 heißen in SQL *Literale*.

Beispiele

In der Tabelle 5.8 sind beispielhaft Literale für die grundlegenden Datentypen angegeben.

Datentyp	Beispiele	Erläuterung
alpha-numerisch	`'Gauss'` `'Gauss''sche Gleichung'`	Die Doppelung des einfachen Anführungszeichens führt dazu, dass es einfach in das Ergebnis aufgenommen wird.

numerisch	`123`
	`123.12`
	`+123`
	`-123.12`
	`123.12E+2`
	`123.12E-2`
		Als Dezimaltrennzeichen ist der im Englischen übliche Punkt zu verwenden.
		Führende Vorzeichen sind erlaubt.
		Die wissenschaftliche Schreibweise ist erlaubt, E+2 steht für 10^2.

Datum/ Uhrzeit	`DATE '2007-08-11'`
	`TIME '13:15:04'`
	`TIMESTAMP '2007-08-11 13:15:04 +1:00'`
		Bei Datums- und Uhrzeitliteralen muss gegebenenfalls die Schreibweise geprüft werden. Hier können sowohl Datenbankeinstellungen als auch Betriebssystemwerte ein anderes Format erfordern.

Tabelle 5.8
Beispiel für Literale

Alphanumerische Literale

Alphanumerisch

Grundsätzlich werden Literale in einfache Anführungszeichen eingeschlossen – auf der deutschen Tastatur auf der #-Taste, nicht zu verwechseln mit den Accents. Soll in einem Literal selbst ein solches Anführungszeichen verwendet werden, wird es durch ein vorangestelltes ebenfalls einfaches Anführungszeichen maskiert.

Escape-Zeichen

MySQL verwendet in Literalen sogenannte Escape-Zeichen, um Sonderzeichen einfügen zu können. Dieses Vorgehen und auch die Zeichen entsprechen den aus der Linux-Welt bekannten Zeichen. In sind einige der wichtigsten Zeichen aufgelistet.

Zeichen	Erläuterung
`\'`	Einfaches Anführungszeichen
`\"`	Doppelte Anführungszeichen
`\\`	Backslash
`\n`	Neue Zeile
`\t`	Tabulatorzeichen

Tabelle 5.9
Escape-Zeichen können in MySQL in Literalen verwendet werden.

Numerische Literale

Numerisch

Sollen numerische Literale, oder einfacher ausgedrückt Zahlen, verwendet werden, treten im Allgemeinen keine größeren Schwierigkeiten auf. Zahlen ohne Dezimaltrennzeichen werden als ganzzahlig (`INTEGER`, `INT`, ...), solche mit Dezimaltrennzeichen als Gleitkommazahlen (`FLOAT`, `DOUBLE`, ...) interpretiert.

Das Dezimaltrennzeichen folgt der allgemeinen Logik einer englischen Umgebung. Hier ist das Dezimaltrennzeichen der Punkt „.“ und das Tausendertrennzeichen das Komma „,“. In einer deutschen Umgebung ist es genau umgekehrt, also Komma als Dezimaltrennzeichen und Punkt als Tausendertrennzeichen. Normalerweise gilt bei allen SQL-Interpretern die englische Logik.

Eine Ausnahme bildet MS Access, das für „interessante“ Verwirrungen sorgen kann, wenn teilweise auf die Betriebssystemumgebung, hier Windows, zurückgegriffen wird. Ändern Sie in der Datenbank `tbDozent` einmal testweise den Wert für einen Stundensatz und notieren sich vorher den alten Wert.

Geben Sie „23,45“ ein, dann finden Sie anschließend in der Anzeige die Eingabe „23,45“, also deutsche Schreibweise. Diese Vermutung wird bestätigt, wenn Sie testweise „23.45“ eingeben, was direkt in „2345,00“ umgesetzt wird. Hier scheint Access, das deutsche Format zu verwenden. Geben Sie wieder „23,45“ ein und testen Sie den Wert mit einer Abfrage. Die `SELECT`-Anweisung, um diesen Datensatz abzufragen, funktioniert aber wie in Listing 5.7 zu sehen in englischer Notation. Dieselbe Abfrage mit „23,45“ führt zu einer Fehlermeldung.

Listing 5.7 Test auf die Schreibweise der Zahl

```
SELECT *
FROM tbDozent
WHERE Stundensatz = 23.45;
```

Intern greift der SQL-Interpreter also auf die englische Schreibweise zurück. Gehen Sie davon aus, dass das praktisch immer der Fall ist, und schreiben Sie Ihre SQL-Anweisungen unbeeinflusst von der deutschen Oberfläche auch hier mit der englischen Syntax.

Vorzeichen können angegeben werden, also +123.45 oder -123.45. Zahlen mit fehlendem Vorzeichen werden als positive Zahlen interpretiert.

Die Verwendung der wissenschaftlichen Schreibweise mit Zehnerexponent, also 123.12E+2 oder 123.12e+2 für $123.12 * 10^2 = 12312$, ist ebenso erlaubt wie üblich. Im Übrigen sind diverse datenbankspezifische Erweiterungen vorhanden, die Sie aber der Dokumentation Ihres Datenbanksystems entnehmen sollten.

Datum/Zeit-Literale

Datum/Zeit

Die meisten Probleme ergeben sich erfahrungsgemäß bei den Datums- und Zeitliteralen. Der SQL-Standard ist hier recht eindeutig.

DATE ’jjjj-mm-tt’

TIME ’hh:mm:ss [{+|-}hh:mm]’

TIMESTAMP ’ jjjj-mm-tt hh:mm:ss[.ddddddddd] [{+|-}hh:mm]’

Das Datum (DATE) wird also mit vierstelliger Jahresangabe und jeweils zweistelliger Monats- und Tagesangabe getrennt mit Bindestrichen angegeben. Dieses Format ist sehr sortierfreundlich, da es sich direkt für jede Sortierung nach auf- oder absteigendem Datum eignet. Eine gültige Datumsangabe ist also beispielsweise `'2008-05-17'`. Die Uhrzeit wird jeweils zweistellig, wie bereits von dem entsprechenden Datentyp bekannt, eingegeben.

Der TIMESTAMP kann zusätzlich Bruchteile von Sekunden und eine Angabe der Zeitzone beinhalten, also etwa

```
'2008-07-19 20:12:45.123 +01:00'
```

für eine Angabe einer Zeit in Deutschland. Das funktioniert in allen Datenbanksystemen. Probieren Sie einfach mal

```
SELECT '2008-07-19 20:12:45.123 +01:00' FROM tbDozent;
```

aus. Alle anderen Formate sind zumeist nur Darstellungsformate, die aber bei Eingaben durchaus zu verwenden sind.

5.4 Ausdrücke

Bisher haben wir immer direkt mit Datenfeldern oder mit Literalen gearbeitet, wenn es darum ging, welche Werte als Ergebnis einer SELECT-Anweisung angezeigt werden, wie Tabellen miteinander verbunden wurden, welche Datensätze gefiltert werden und wonach sortiert wird. Betrachten Sie die Syntax der SELECT-Anweisung, haben wir also bisher die folgende Form verwendet:

SELECT [DISTINCT|ALL] *feldnamenliste*

FROM *tabelle* {*joinliste*}

[WHERE *bedingungsliste]*

[GROUP BY *feldnamenliste]*

[HAVING *bedingungsliste]*

[ORDER BY {*feldnamenliste [ASC|DESC]*}*];*

Ausdruck

Tatsächlich kann an (fast) jeder Stelle, an der ein Feldname steht, auch ein Literal stehen. Werden Literale mit Feldnamen oder Literale mit Literalen oder Feldnamen mit Feldnamen über Operatoren wie beispielsweise +, –, * oder / miteinander verbunden, entsteht ein sogenannter Ausdruck. Jeder einzelne Feldname für sich, wie auch jedes Literal für sich ist aber ebenfalls bereits ein Ausdruck. Ein Ausdruck kann also ein Feldname sein, ein Literal oder eine Kombination aus beiden in beliebiger Komplexität.

Zusätzlich stehen bei der Bildung von Ausdrücken sogenannte Funktionen zur Verfügung, mit denen die Feldinhalte und Literale weiter verändert werden können. Somit stellt der Begriff „Ausdruck“ hier einen Oberbegriff für alle Kombinationen von Feldern, Funktionen und Operatoren dar, die zu einem sinnvollen Ergebnis führen.

Beispiele

Die folgende SELECT-Anweisung gibt die Kursgebühr um 19 % Umsatzsteuer erhöht und auf zwei Stellen gerundet aus. Dabei wird die Währung „EUR“ als Text angefügt.

Listing 5.8 Kalkulation des Bruttobetrages

```
SELECT
   k.Kurskennung,
   CONCAT(ROUND(k.Gebuehr*1.19,2),' EUR') AS "Gebühr"
FROM tbKurs k;
```

Hier sind die beiden Feldnamen Kurskennung und Gebuehr, das Literal ' EUR' (beachten Sie das Leerzeichen vor dem „EUR“) sowie die Funktionen ROUND() und CONCAT() verwendet worden. Die SELECT-Anweisung kann somit erweitert werden zu:

SELECT [DISTINCT|ALL] *ausdrucksliste*

FROM *tabelle* {*joinliste*}

[WHERE *bedingungsliste]*

[GROUP BY *ausdrucksliste]*

[HAVING *bedingungsliste]*

[ORDER BY {*ausdrucksliste [ASC|DESC]*}*];*

5.5 Funktionen

Funktionen bilden einen wichtigen Bestandteil vieler Ausdrücke. Teilweise werden ganze Datenbanksysteme nach dem Umfang der bereitgestellten Funktionen beurteilt, was aber zumindest zweifelhaft ist.

Es gibt zwei große Gruppen von Funktionen:

- Datensatzorientierte Funktionen, die Berechnungen innerhalb eines Datensatzes ausführen, um neue Ausdrücke zu berechnen (Skalarfunktionen).
- Gruppenorientierte Funktionen, die basierend auf einer Gruppe (Menge) von Datensätzen Berechnungen innerhalb eines Datenfeldes für die gesamte Gruppe durchführen. (Aggregatfunktionen).

Datensatzorientierte Funktionen erlauben die Berechnung neuer Ausdrücke, also neuer Spalten, innerhalb der Ergebnismenge einer SELECT-Anweisung. Damit entstehen in der Abfrage Spalten, die in keiner Tabelle der Datenbank vorhanden sind, wie in unserem obigen Beispiel die Gebühr.

Diese Funktionen haben alle einen festgelegten Aufbau, der den Funktionen in anderen Programmiersprachen entspricht:

FUNKTIONSNAME (Parameter1, Parameter2, ...)

Parameter

Der Funktionsname wird hier stets in Großbuchstaben geschrieben, obwohl dies nicht zwingend notwendig ist. Er ist aber ähnlich wie die SQL-Schlüsselwörter fest definiert und wird daher hier genauso behandelt. In den Klam-

mern stehen die Parameter, also die Werte, die an die Funktion übergeben werden. Wichtig ist dabei, dass sowohl die Anzahl als auch der Datentyp der Parameter mit der Funktionsdefinition übereinstimmen.

Freiheiten

SQL ist allerdings recht gutmütig, was den Datentyp angeht. Zumeist reicht es hier, die Kategorien alphanumerisch, numerisch sowie die verschiedenen Datums- und Uhrzeitformate beziehungsweise Intervalle und `BOOLEAN` (Wahrheitswerte) zu unterscheiden. Bei der Anzahl der Parameter gibt es teilweise auch Freiräume, insofern, als dass teilweise weiter hinten stehende Parameter optional sind und bei Fehlen einer Angabe diese Werte durch Standardwerte ersetzt werden.

Rückgabewert

Die Parameter sind reine Eingangswerte, sie können nicht geändert werden. Im Sinne der Programmierung handelt es sich also bei SQL immer um einen „Call By Value“: Werte können übergeben, aber nicht verändert werden. Jede Funktion liefert immer genau einen Rückgabewert, der bei der Ausführung an die Stelle des Funktionsnamens tritt. Daher ist der Datentyp des Ergebnisses, das eine Funktion liefert, von zentraler Bedeutung für ihre Verwendung. Sie darf nur dort verwendet werden, wo der Datentyp, den sie liefert, verwendet werden darf.

Der Datentyp, den eine Funktion als Ergebnis liefert, ist oft derselbe den auch der zentrale Eingabeparameter hat. Das ist aber nicht zwingend erforderlich. Daher lassen sich die Funktionen nur grob nach den drei wichtigsten Datentypklassen und in zwei weitere Gruppen einteilen.

- Numerische Funktionen, die auf der Umformung und Berechnung von numerischen Werten beruhen – also weitgehend mathematische Funktionen, mit denen sich beispielsweise aus Bruttobeträgen die Nettobeträge oder die Umsatzsteuer berechnen lässt oder mit denen sich der Gewinn aus vorhandenen Datenfeldern bestimmen lässt.
- Alphanumerische Funktionen, die sich auf alle alphanumerischen Datentypen wie `CHAR` oder `VARCHAR` anwenden lassen und Texte oder einzelne Zeichen aus anderen Texten extrahieren, Texte miteinander verbinden, Zeichen ersetzen oder löschen. Damit lassen sich beispielsweise Adressen aus ihren Einzelteilen neu kombinieren oder umgekehrt in längeren Textteilen Teilbegriffe oder Worte suchen.
- Datumsfunktionen, die im Wesentlichen auf `DATE`, `TIME`, `TIMESTAMP`, aber auch auf Zeitintervallen beruhen, um Zeiten zu vergleichen, Zeiträume zu berechnen oder auch nur die aktuelle Zeit zu ergänzen.
- Casting-Funktionen, die im Zweifelsfall Werte zwischen Datentypen umwandeln, soweit dies sinnvoll und aufgrund der Werte machbar ist.
- Sonstige Funktionen, die beispielsweise einen logischen Ablauf erlauben, Systemvariablen abfragen oder sonstige Informationen liefern.

Einteilung nach dem Ergebnisdatentyp

Beachten Sie, dass die Einteilung im SQL-Standard dem Datentyp folgt, den die Funktion erzeugt, nicht dem, den sie primär als Parameter verarbeitet. So gilt die Ermittlung der Länge einer Zeichenkette als numerische Funktion, weil sie zwar eine Zeichenkette, also eine alphanumerische Eingabe, besitzt,

das Ergebnis aber immer eine Zahl, also eine numerische Angabe, ist. Für jede Funktion sind die benötigten Eingabewerte (Parameter) und deren Datentypen festgelegt, und die Logik der SQL-Anweisung muss sicherstellen, dass genau diese Datentypen auch für die Funktion bereitgestellt werden.

Für die eigene Erstellung von SQL-Anweisungen, insbesondere SELECT-Anweisungen, ist diese Einteilung aber oft auch lästig, da man zumeist von bestimmten Datenfeldern ausgeht, deren Datentyp man kennt. Man weiß, dass man beispielsweise zwei alphanumerische Felder verbinden und deren gemeinsame Länge bestimmen will. Da ist es eher ungewöhnlich, bei den numerischen Funktionen zu suchen, weil am Ende eine Zahl erzeugt wird. Daher sind die Funktionen hier nach ihrem primären Anwendungsbereich, also nach ihrer Nutzung im Zusammenhang mit numerischen, alphanumerischen beziehungsweise datumsorientierten Ausdrücken, sortiert.

Unterschiedlicher Funktionsumfang

Bevor wir jetzt auf die Funktionen eingehen, muss noch eine kleine Anmerkung erfolgen. Der Standard gab ursprünglich sehr wenig hinsichtlich der Funktionen vor und ist der Entwicklung der Datenbanksysteme immer hinterhergelaufen. Daher gibt es zwar eine ähnliche Funktionalität und auch eine recht große gemeinsame Menge an Funktionen bei den verschiedenen Systemen, aber auch eine Fülle von Funktionen, die nur in bestimmten Systemen existieren und hier nicht alle vorgestellt werden können.

Die gute Nachricht ist, dass die Datenbankhersteller bemüht sind, den Anwendern den Umstieg zu erleichtern, und viele Funktionen gleich benennen, manchmal sogar dieselbe Funktion unter verschiedenen Namen anbieten, um die Kompatibilität zu verbessern.

Als kleines Beispiel kann dafür die Funktion CURRENT_TIMESTAMP() dienen, die das aktuelle Systemdatum und die Systemzeit im TIMESTAMP-Format liefert. In MySQL können Sie den aktuellen TIMESTAMP auf mindestens drei Arten erstellen. Die folgende SELECT-Anweisung liefert dreimal dasselbe Ergebnis:

```
SELECT NOW(), SYSDATE(), CURRENT_DATE;
```

Bevor wir uns den Funktionen im Detail widmen, noch ein paar Hinweise zu einigen Datenbanken, um keinen zu großen Frust aufkommen zu lassen.

MySQL

MySQL bietet einen großen Funktionsumfang mit teils historisch gewachsenen und teils in neuerer Zeit stärker am Standard und anderen Systemen orientierten Funktionen. Sie finden sowohl in der PDF-Datei *refman-5.1-de.a4.pdf* als auch in der Oberfläche im unteren rechten Fenster unter dem zweiten Reiter FUNKTIONEN eine umfangreiche Dokumentation.

Oracle

Oracle ist als die Datenbank mit den meisten und mächtigsten Funktionen bekannt. Sie finden die Dokumentation am besten online, indem Sie im Übersichtsfenster (HOME) rechts oben auf LINKS/DOCUMENTATION klicken. Geben Sie beispielsweise „SQL function“ ein und wählen Sie einen Eintrag, der erste ist oft eine gute Wahl.

Firebird UDF

Firebird verfolgt ein etwas anderes Konzept als andere Datenbanken. Hier wird nur ein relativ kleiner Befehlssatz im Standard angeboten. Dafür wird die Möglichkeit angeboten, sogenannte UDF (User Defined Functions) in einer Programmiersprache zu erstellen und dann einzubinden. Derartige Bibliotheken sind verfügbar, die Einbindung soll aber den Spezialisten für Firebird überlassen werden. Hier wird nur der Standard verwendet. Informationen finden Sie im Internet am besten unter *IBPhoenix.com* oder über eine entsprechende Suche. Die Interbase-Dokumentation ist auch noch hilfreich.

MS Access

Die Access-Funktionen können Sie natürlich in der Hilfe nachschlagen, allerdings kann das etwas unübersichtlich werden, wenn Sie nicht wissen, wie die Funktion heißt. Online finden Sie Hilfe bei Microsoft unter *office.microsoft.com/de-ch/access/HA101316761031.aspx*.

openBase

Für openBase finden Sie neben der Hilfe ebenfalls im Internet einige Hinweise, so beispielsweise unter *www.hsqldb.org/doc/guide/ch09.html* zum Thema SQL-Syntax und SQL-Funktionen.

Wie immer sind die Informationen Momentaufnahmen der jeweiligen Versionen und Releases. Jetzt werden einige wichtige Funktionen besprochen.

5.6 Datensatzorientierte Funktionen (Skalarfunktionen)

Im Folgenden werden immer zunächst die Funktionen mit ihrer Funktionsweise aufgelistet. Dann werden die Namen in den einzelnen Systemen angegeben und einige Beispiele ergänzt.

Die Beschreibung entspricht der folgenden Legende.

Legende

Die Eingabeparameter werden entsprechend des Datentyps angegeben.

`n`, `n2`, `n3`numerische Angaben

`a`, `a2`, `a3`alphanumerische Angaben

`d`, `d2`, `d3`Datumsangaben

`u`, `u2`, `u3`Uhrzeitangaben

`t`, `t2`, `t3`Timestamp-Angaben

`i`, `i2`, `i3`Zeitintervall-Angaben

`[Parameter]` optionaler Parameter

Das Ergebnis wird mit einem Pfeil angegeben, beispielsweise eine Funktion, die einen numerischen Wert liefert: `Funktion() -> n`

5.6.1 Funktionen in MS Access

In MS Access stellen die Funktionen ein besonderes Kapitel dar. Um für die MS Access-Anwender die Übersicht etwas zu vereinfachen, soll hier vor der eigentlichen Funktionsbeschreibung auf die Besonderheiten hingewiesen werden.

Jet Engine

Sie müssen hier zwischen den Funktionen der Jet Engine und den Funktionen der MS Access-Oberfläche unterscheiden. Die Jet Engine verfügt weitgehend über die beispielsweise auch in MySQL bekannten Funktionen, wie sie später ausführlich beschrieben werden. Je nach Betriebsart muss aber unterschieden werden, ob im ODBC-Modus oder im DAO-Modus (dann zumeist mit dem SQL Server) als sogenanntes Access-Projekt (ADP) gearbeitet wird.

ODBC

Im ODBC-Modus stehen die Funktionen in Tabelle 5.10 zur Verfügung. Wenn Sie diese nutzen, können Sie später, wenn wir die Funktionen beschreiben, zumeist leicht aus dieser Tabelle die passende Funktion ermitteln, soweit sie verfügbar ist.

Tabelle 5.10 ODBC-Skalarfunktionen der MS Access Jet Engine

Funktionsklasse	Funktionen
numerisch	ABS, ATAN, CEILING, COS, EXP, FLOOR, LOG, MOD, POWER, RAND, SIGN, SIN, SQRT, TAN
alphanumerisch	ASCII, CHAR, CONCAT, LCASE, LEFT, LENGTH, LOCATE, LTRIM, RIGHT, RTRIM, SPACE, SUBSTRING, UCASE
Datum/Uhrzeit	CURDATE, CURTIME, DAYNAME, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, HOUR, MINUTE, MONTH, MONTHNAME, NOW, QUARTER, SECOND, WEEK, YEAR
Konvertierung	CONVERT

ADP

Wird dagegen in einem ADP gearbeitet, stehen die Funktionen des MS SQL Servers zur Verfügung, die dort nachzulesen sind. Die Jet Engine wird bei der Programmierung über die entsprechende Schnittstelle genutzt, also im Normalfall nicht, wenn wir wie in diesem Buch üblich mit der MS Access-Oberfläche arbeiten.

VBA

Bei der Nutzung der MS Access-Oberfläche wird dagegen, wie wir es hier tun, auf den VBA-Funktionssatz zurückgegriffen. Dies sind VBA-Funktionen, die für das Access-Modul verfügbar gemacht worden sind und in SQL verwendet werden dürfen. Diese können sowohl direkt im SQL-Code des Abfragefensters verwendet werden als auch über die grafische Oberfläche eingegeben werden.

Bei der Nutzung der grafischen Oberfläche (Entwurfsansicht für Abfragen) klicken Sie mit der rechten Maustaste auf die zu beschreibende Spalte im Abfragefenster. Es erscheint ein Auswahlmenü wie in Abbildung 5.7 angegeben.

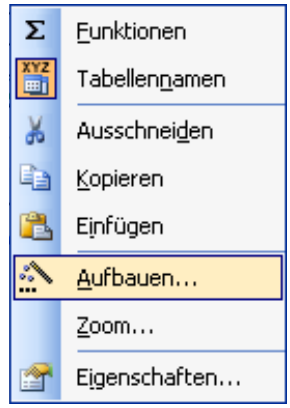


Abbildung 5.7
Auswahl nach Klick mit rechter Maustaste auf eine Spalte

Wählen Sie jetzt die Option AUFBAUEN, um in den Ausdrucks-Generator zu kommen (siehe Abbildung 5.8). Der Ausdrucks-Generator bietet unter FUNKTIONEN/EINGEBAUTE FUNKTIONEN alle VBA-Funktionen an, die zur Verfügung stehen. Diese sind in der Oberfläche thematisch geordnet. Bei Auswahl einer Funktion wird außerdem unten im Fenster angezeigt, wie viele Parameter die Funktion benötigt und welchen Datentyp diese haben müssen, hier `Abs(»Zahl«)`.

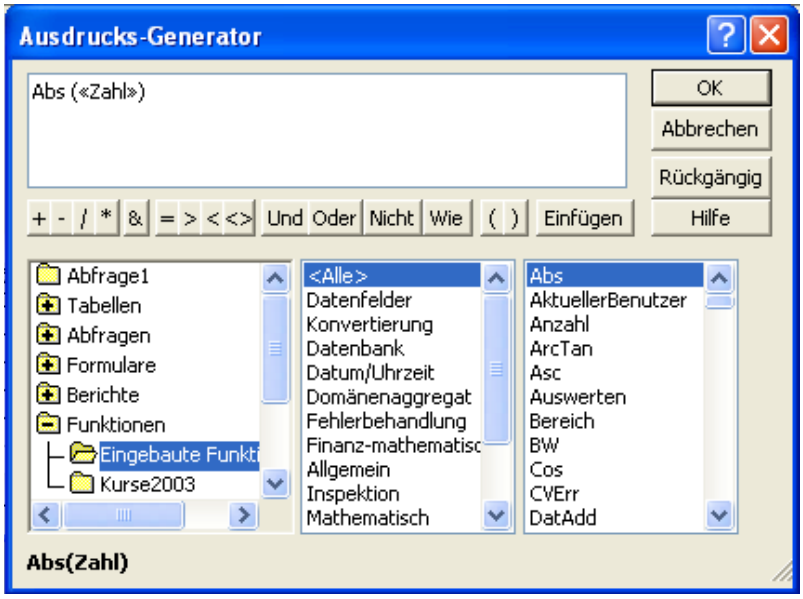


Abbildung 5.8
Ausdrucks-Generator für die Nutzung von Funktionen

Um die Sache jetzt nicht zu einfach zu machen, sind diese Funktionen ins Deutsche übersetzt worden, während im SQL-Fenster die englischen Bezeichnungen einzugeben sind. So finden Sie beispielsweise im Ausdrucks-Generator die Funktion RUNDEN (ab Access2003), muss aber im SQL-Fenster `ROUND()` eingeben. Daher ist in den folgenden Tabellen bei der Angabe der

Funktionsnamen für MS Access immer zunächst der englische Name, wie wir ihn in SQL benötigen, und danach der deutsche Name des Ausdrucks-Generators angegeben.

Die Handhabung des Ausdrucks-Generators können Sie der MS Access-Hilfe entnehmen. Hier soll SQL im Vordergrund stehen, daher wird im Folgenden auch bei MS Access immer nur das Beispiel im SQL-Fenster angegeben. Hier verhält sich MS Access dann glücklicherweise wieder recht standardkonform.

5.6.2 Numerische Funktionen

Die einfachsten und zugleich auch oft die wichtigsten Funktionen stellen die mathematischen Funktionen dar.

Beispiel

In der Tabelle `tbKurs` sind die Kursgebühren für die einzelnen Kurse aufgeführt. Dabei handelt es sich um Nettopreise. Jetzt sollen für alle Kurse, die umsatzsteuerpflichtig sind, neben den Nettopreisen auch die Umsatzsteuer (19 %) und der sich dann ergebende Bruttopreis in der `SELECT`-Anweisung angegeben werden. Die `SELECT`-Anweisung hierzu lautet dann:

Listing 5.9
Mathematische Operatoren zur Berechnung von Ausdrücken

```
SELECT
   k.kurskennung,
   k.ustpflichtig,
   k.Gebuehr AS "Gebuehr",
   k.Gebuehr * 0.19 AS "Umsatzsteuer",
   k.Gebuehr * 1.19 AS "Gebühr brutto"
FROM tbKurs k
WHERE (k.ustpflichtig='J');
```

Das Ergebnis ist in Abbildung 5.9 dargestellt.

Abbildung 5.9
Ergebnis der SELECT-Anweisung mk Umsatzsteuer

KURSKENNUNG	USTPFLICHTIG	Gebuehr	Umsatzsteuer	Gebühr Brutto
Celle17-Word	J	280	53,2	333,2
Celle20-Word	J	280	53,2	333,2
Hannover90-Windows	J	120	22,8	142,8
Hannover91-Windows	J	120	22,8	142,8

4 rows returned in 0,08 seconds CSV Export

Operatoren +, –, *, /

Hier ist zunächst die Grundrechenart * zum Multiplizieren verwendet worden. Entsprechend funktionieren die anderen Grundrechenarten, sodass +, –, * und / immer als Operatoren für numerische Angaben zur Verfügung stehen. Beachten Sie, dass das „+“- und das „-“-Zeichen zwei Aufgaben übernehmen. Sie können unäre Operatoren (Vorzeichen) sein, also +2 oder -3. Sie können auch binäre Operatoren im Sinne der normalen Grundrechenarten, also 5+3 oder 5-7, sein. Glücklicherweise unterscheidet SQL hier nicht.

DIV, MOD

Zusätzlich werden oft noch Operatoren für die ganzzahlige Division (`DIV`) sowie für die Berechnung des Restes angeboten (`MOD` oder `%`).

Sie können in den meisten Systemen eine SELECT-Anweisung auch als reine Rechenoperation nutzen, beispielsweise

```
SELECT 5-7;
```

was dann eine Spalte mit der Überschrift 5-7 und eine Zeile mit dem Wert „-2“ liefert. Sie können SELECT also auch als recht umständlichen Taschenrechner nutzen. Der SQL-Standard erfordert eigentlich eine FROM-Klausel, die hier komplett fehlt, aber sie wird inhaltlich auch nicht benötigt, sodass MySQL beispielsweise komplett darauf verzichten kann. Andere Datenbanken wie Oracle behelfen sich hier mit einzeiligen Hilfstabellen wie „dual“, sodass man dann

```
SELECT 5-7
FROM dual;
```

einzugeben hat.

Die wichtigsten numerischen Funktionen sind in Tabelle 5.11 zusammengestellt.

Tabelle 5.11 Gängige numerische Funktionen

SQL-Funktion	Erläuterung	Beispiel
`ABSOLUTE(n) -> n`	Bestimmung des absoluten Wertes von n, also Entfernung des Vorzeichens	`ABS(-123) -> 123` `ABS(123) -> 123`

`CEILING(n) -> n`
`CEIL(n) -> n`
	„Zwangsaufrunden“ der Nachkommastellen. Es wird die kleinste ganze Zahl größer oder gleich der Eingabe ermittelt.	`CEILING(12.45) -> 13`
		`CEIL(-12.45) -> -12`
		`CEIL(12) -> 12`

`FLOOR(n) -> n`	„Zwangsabrunden“ der Nachkommastellen. Es wird die größte ganze Zahl kleiner oder gleich der Eingabe ermittelt.	`FLOOR(12.45) -> 12`
		`FLOOR(-12.45) -> -13`
		`FLOOR(12) -> 12`

`EXP(n) -> n`	Ermittelt e^n, also die Eulersche Zahl hoch dem angegebenen Wert	`EXP(1) -> 2,71828`
`POWER(n, n2) -> n`	Ermittelt n^{n2}, also die normale Potenzfunktion	`POWER(2,5) -> 32`
`LEAST(n, n1, n2,...) –n`	Kleinster Wert einer Reihe von Werten, die einzeln aufgezählt als Parameter angegeben werden.	`LEAST(2,3,5) -> 2`

`LN(n) -> n`	Natürlicher Logarithmus (zur Basis e), verbotene Werte <= 0 liefern NULL	`LN(2,71828) -> 1`
		`LN(1) -> 0`
		`LN(-1) -> NULL`

`LOG(n1, n) -> n`	Logarithmus zur Basis n1 von n	`LOG(10,100) -> 2` `LOG(0,100) -> NULL`

Tabelle 5.11 (Forts.) Gängige numerische Funktionen

SQL-Funktion	Erläuterung	Beispiel
`LOG2(n) -> n`	Logarithmus zur Basis 2 von `n`	`LOG2(8) -> 3`
`LOG10(n) -> n`	Logarithmus zur Basis 10 von `n`	`LOG10(100) -> 2`

`MOD(n, n1) -> n`
`n % n1 -> n`
	Bestimmt den Rest der ganzzahligen Division `n / n1`. Teilweise darf auch das Prozentzeichen als Operator verwendet werden.	`MOD(124,10) -> 4`
		`26 % 5 -> 1`
		`27 MOD 9 -> 0`

`PI() -> n`	Liefert den Wert der Zahl Pi.	`PI() -> 3.14159...`
`RANDOM([n])`	Erzeugt eine Zufallszahl zwischen 0 und 1. Wird `n` angegeben, kann eine definierte (wiederholbare) Folge erreicht werden.	`RAND() -> 0,254...` `RAND(1) -> 0,343...`
`GREATEST(n, n1, n2, ...) -> n`	Größter Wert einer Reihe von Werten, die einzeln aufgezählt als Parameter angegeben werden.	`GREATEST(2,3,5) ->5`

`ROUND(n,[n1]) -> n`	Rundet `n` auf die angegebene Anzahl Dezimalstellen `n1`. Fehlt `n1` wird auf ganzzahlige Werte gerundet.	`ROUND(12.45) -> 12`
		`ROUND(12.453, 2) -> 12.45`
		`ROUND(12) -> 12`

`SIGN(n) -> n`	Liefert das Vorzeichen von `n` als Zahlwert
	positiv (1), null (0), negativ (-1)
		`SIGN(5) -> 1`
		`SIGN(0) -> 0`
		`SIGN(-7) -> -1`

`SQRT(n) -> n`	Positive Quadratwurzel von `n`	`SQRT(4) -> 2` `SQRT(-1) -> NULL`
`TRUNCATE(n, [n1]) -> n`	Alle Nachkommastellen werden auf `n1` Stellen abgeschnitten (nicht gerundet). Wird `n1` nicht angegeben, wird auf eine ganze Zahl abgeschnitten.	`TRUNCATE(12.45,0)-> 12` `TRUNCATE(-1,999) -> -2`

Tabelle 5.12 Vergleich numerischer Funktionen verschiedener Datenbanken

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
`ABSOLUTE()`	`ABS()`	`Abs()`	`ABS()`	`%`	`ABS()`
`CEILING()` `CEIL()`	`CEILING()` `CEIL()`	`%`	`CEIL()`	`%`	`CEILING()`
`FLOOR()`	`FLOOR()`, `DIV()`	`Int()`, `Fix()` bei neg. Zahlen wie `FLOOR()`	`FLOOR()`	`%`	`FLOOR()`

Tabelle 5.12 (Forts.) Vergleich numerischer Funktionen verschiedener Datenbanken

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
EXP()	EXP()	Exp() Exponen-tial()	EXP()	%	EXP()
POWER()	POWER()	%	POWER()	%	POWER()
LEAST()	LEAST()	%	LEAST()	%	%
LN()	LN()	Log(x) Logarith-mus()	LN()	%	%
LOG()	LOG()	%	LOG()	%	LOG()
LOG2()	LOG2()	%	LOG(2,)	%	LOG(2,)
LOG10()	LOG10()	%	LOG(10,)	%	LOG10()
MOD()	MOD(), X % Y, X MOD Y	%	MOD()	%	MOD()
PI()	PI()	%	%	%	PI()
RANDOM()	RAND()	Rnd(), ZZG()	dbms_ random. value()	%	RAND()
GREA-TEST()	GREA-TEST()	%	GREA-TEST()	%	%
ROUND()	ROUND()	Round(), Runden()	ROUND()	%	ROUND()
SIGN()	SIGN()	Sgn(), Vorzchn()	SIGN()	%	SIGN()
SQRT()	SQRT()	Sqr(), QWurzel()	SQRT()	%	SQRT()
TRUN-CATE()	TRUN-CATE()	%	TRUNC()	%	TRUNCATE()

Daneben existieren noch die zumeist selten gebrauchten trigonometrischem Funktionen und Winkelumrechnungen wie `ACOS(X)`, `ASIN(X)`, `ATAN(X)` (in verschiedenen Varianten), `COS(X)`, `COT(X)`, `SIN(X)`, `TAN(X)` sowie `DEGREES(X)`, `RADIANS(X)`.

Beispiel

Betrachten wir den Aufruf und die Funktion an einem Beispiel. Es gibt verschiedene „Rundungsfunktionen“, `ROUND`, `CEIL` und `FLOOR`, die im Wesentlichen ein mathematisches Runden, ein Aufrunden und ein Abrunden bewirken. In Listing 5.10 sind alle drei Varianten verwendet worden.

Listing 5.10 Runden mit verschiedenen Funktionen

```
SELECT
   Gebuehr,
   KursdauerStunden,
   Gebuehr/KursdauerStunden,
   ROUND(Gebuehr/KursdauerStunden,2) AS "gerundet",
   CEIL(Gebuehr/KursdauerStunden) AS "aufgerundet",
   FLOOR(Gebuehr/KursdauerStunden) AS "abgerundet"
FROM tbKurs t;
```

Bei der mathematischen Rundung ist ein zweiter Parameter angegeben worden, der die Anzahl der Nachkommastellen angibt. Dadurch wird hier nicht auf ganze Zahlen, sondern auf die zweite Nachkommastelle, also auf Hundertstel, gerundet. Die Eingabe der Parameter mit runden Klammern ist in allen Systemen gleich. Das Ergebnis der Abfrage ist in Abbildung 5.10 zu sehen.

Abbildung 5.10
Ergebnis der Abfrage mit Rundungen

Gebuehr	KursdauerStun...	Gebuehr/Kursd...	gerundet	aufgerundet	abgerundet
280	40	7	7.00	7	7
280	40	7	7.00	7	7
350	36	9.7222222222...	9.72	10	9
400	40	10	10.00	10	10
350	32	10.9375	10.94	11	10
120	18	6.66666666666...	6.67	7	6
120	18	6.66666666666...	6.67	7	6

Beispiel

Sollen beispielsweise die Kursstunden pro Tag auf 7 Stunden reduziert werden, so stellt sich die Frage, wie viele Tage dann ein Kurs benötigt und wie viele Stunden übrig bleiben, die keinen kompletten Tag füllen. Diese müssen entweder entfallen oder es müssen zusätzliche Stunden eingeplant werden, damit ein weiterer Tag gefüllt werden kann. Einen Überblick können Sie sich mit einem Befehl wie in Listing 5.11 (MySQL) verschaffen.

Listing 5.11
Überblick Kursaufteilung bei siebenstündigem Unterricht

```
SELECT
   DauerPlan,
   DauerPlan DIV 7 AS "Ganze Tage",
   MOD(DauerPlan,7) AS "Reststunden",
   7 - MOD(DauerPlan,7) AS "Aufzufüllende Stunden"
FROM tbKursthema;
```

Das Ergebnis ist in Abbildung 5.11 zu sehen.

Abbildung 5.11
Ergebnis der Abfrage aus Listing 5.11

DauerPlan	Ganze Tage	Reststunden	Aufzufüllende Stunden
20	2	6	1
20	2	6	1
8	1	1	6
40	5	5	2
40	5	5	2
40	5	5	2
40	5	5	2
80	11	3	4
80	11	3	4
40	5	5	2
40	5	5	2

Info

Mit SQL2003 wurde eine Reihe von Funktionen neu in den Standard übernommen. Dazu gehören der natürliche Logarithmus `LN()` sowie die e-Funktion `EXP()`, die normale Potenzfunktion `POWER()`, die Quadratwurzel `SQRT()`, das Abschneiden von Nachkommastellen `FLOOR()`, die Bestimmung der nächstgrößeren ganzen Zahl `CEIL[ING]()` sowie `WIDTH_BUCKET(,,,)`, um Daten zu gruppieren. Diese Funktionen, die nicht sonderlich ungewöhnlich sind und zumindest teilweise bereits in vielen Systemen vorher implementiert waren, zeigen wieder, dass der Standard der Entwicklung in vielen Bereichen mit Abstand folgt.

Abschließend soll noch eine neuere komplexe Funktion gezeigt werden, die die Klassifizierung von Daten erlaubt: `WIDTH_BUCKET`. Da diese Funktion nur in Oracle vollständig implementiert ist, werden die Möglichkeiten dieser neuen Gruppierungsfunktion an einem Beispiel mit Oracle gezeigt.

Listing 5.12
Gruppierung der bezahlten Kursbesuche

```
SELECT
   WIDTH_BUCKET(GezahlterBetrag, 0, 400, 8) AS "Klasse",
   COUNT(1) AS "Anzahl Kursbesuche"
FROM tbKursbesuche t
GROUP BY WIDTH_BUCKET(GezahlterBetrag,0, 400, 8)
ORDER BY WIDTH_BUCKET(GezahlterBetrag,0, 400, 8);
```

Dabei werden die Gebühren, die von den einzelnen Kursteilnehmern bereits bezahlt wurden, aus dem Feld `GezahlterBetrag` gelesen. Mit der Funktion `WIDTH_BUCKET` werden die Daten in diesem Feld klassifiziert. Der niedrigste Betrag ist 0, der höchste 400 Euro. Dieser Bereich wird in 8 gleich große Klassen aufgeteilt. `WIDTH_BUCKET` ermittelt für jeden Wert, zu welcher Klasse er gehört. Gruppiert und sortiert nach dieser Klassennummer erhält man das Ergebnis in Abbildung 5.12.

Abbildung 5.12
Klassifizierte Kursbeiträge

Klasse	Anzahl Kursbesuche
1	2
2	3
3	1
6	5
7	2
8	5

Übungen zu den numerischen Funktionen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Erstellen Sie eine Übersicht über alle Kursbesuche mit der `KTID`, der Gebühr für den Kurs und dem Rabatt. Berechnen Sie außerdem eine Spalte Reduzierte Gebühr, die den für den Kurs zu zahlenden Beitrag bestimmt. (Ü5.6.2.1)
2. Bestimmen Sie für alle Kurse die Gebühr, die Kosten für den Dozenten, sowie die Anzahl der Teilnehmer, die man benötigt, um die Kosten zu decken, wenn neben den Kosten für den Dozenten noch einmal der gleiche Betrag für sonstige Kosten anfällt. Geben Sie zusätzlich die Anzahl aufgerundet auf ganze Teilnehmer an. (Ü5.6.2.2)

3. Nutzen Sie die KTHID der Kursthemen, um den Wert der Exponentialfunktion sowie die Quadratwurzel der ersten 10 Zahlen zu bestimmen. (Ü5.6.2.3)
4. Nutzen Sie die Kursbesuche, um 18-mal zu „würfeln“, also eine ganze Zufallszahl im Bereich von „1“ bis „6“ zu erzeugen. (Ü5.6.2.4)

5.6.3 Alphanumerische Funktionen

Die zweite Gruppe von Skalarfunktionen sind Funktionen, die sich auf alphanumerische Datentypen anwenden lassen, also auf Texte. Einige wichtige alphanumerische Funktionen sind in Tabelle 5.13 zusammengestellt.

Tabelle 5.13 Gängige alphanumerische Funktionen

SQL-Funktion	Erläuterung	Beispiel
`CONCATENATE (a||a2) -> a` oder `CONCATENATE(a,a2) -> a`	Die beiden Texte a und a2 werden aneinandergehängt.	`CONCATENATE ('29223', ' Celle') -> '29223 Celle'`
`ASCII(a) -> n`	ASCII-Zeichencode des Zeichens a. a muss ein einzelnes Zeichen sein.	`ASCII('A') -> 65`
`CHAR(n) -> a`	Liefert das Zeichen, das zu dem angegebenen ASCII-Code gehört.	`CHAR(65) -> 'A'`
`POSITION (a IN a2) -> n`	Sucht das erste Vorkommen des Textes a im Text a2. Oft wird statt des IN auch ein normales Komma verwendet. Die Zeichen werden ab 1 gezählt.	`POSITION ('an' IN 'Hannover') -> 2`
`LENGTH(a) -> n`	Gibt die Anzahl der Zeichen n im Text a an.	`LENGTH('Hallo') -> 5`
`LOWER(a) -> a`	Wandelt alle Zeichen des Textes a in Kleinbuchstaben um und gibt diesen Text aus.	`LOWER('Name') -> 'name'`
`SUBSTRING(a,n,n2) -> a`	Ermittelt den Teil von a als Text, der beim n-ten Zeichen anfängt und die Länge n2 hat.	`SUBSTRING ('Hannover', 2, 4) -> 'anno'`
`REPLACE(a,a2,a3) -> a`	In a wird das Auftreten von a2 durch a3 ersetzt.	`REPLACE('Tisch', 'T', 'F') -> 'Fisch'`
`TRIM(a) -> a`	Entfernt aus dem Text a alle führenden und alle anhängenden Leerzeichen.	`TRIM (' Text ') -> 'Text'`

Tabelle 5.13 (Forts.)
Gängige alphanumerische Funktionen

SQL-Funktion	Erläuterung	Beispiel
LTRIM(a) -> a	Entfernt aus a alle führenden Leerzeichen.	LTRIM (' Text') -> 'Text '
RTRIM(a) -> a	Entfernt aus a alle anhängenden Leerzeichen.	RTRIM(' Text ') -> ' Text'
UPPER(a) -> a	Wandelt den Text a in Großbuchstaben um.	UPPER('name' -> 'NAME'

Tabelle 5.14
Alphanumerische Funktionen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
CONCATENATE()	CONCAT()	text1 & text2	CONCAT()	text1 || text2	CONCAT()
ASCII()	ASCII()	Asc()	ASCII()	%	ASCII()
CHAR() CHAR()		Chr(), Zchn()	CHR()	%	CHAR()
POSITION()	POSITION (a IN a2), INSTR (Basis, Suchtext) LOCATE Suchtext, Basis)	InStr (Basis, Suchtext)	INSTR (Basis, Suchtext)	%	POSITION (..IN..) LOCATE (Suchtext, Basis)
LENGTH()	LENGTH()	Len() Länge()	LENGTH()	%	LENGTH()
LOWER()	LOWER()	Lcase() Kleinbst()	LOWER()	%	LOWER() LCASE()
SUBSTRING ()	SUBSTRING () MID()	Mid() Teil()	SUBSTR()	SUBSTRING (wert FROM .. FOR ..)	SUBSTRING (..FROM .. FOR ..) SUBSTR()
REPLACE()	REPLACE()	Replace() Ersetzen()	REPLACE()	% REPLACE()	
TRIM()	TRIM()	Trim() Glätten()	TRIM()	TRIM (FROM wert)	%
LTRIM()	LTRIM()	Ltrim() LGlätten()	LTRIM()	TRIM (LEADING FROM wert)	LTRIM()
RTRIM()	RTRIM()	Rtrim() RGlätten()	RTRIM()	TRIM (TRAILING FROM wert)	RTRIM()
UPPER()	UPPER()	Ucase() Grossbst()	UPPER()	UPPER()	UPPER() UCASE()

Beispiel

In der Tabelle `tbKursthema` sind die Kursbeschreibungen für die einzelnen Kursthemen enthalten. Jetzt soll für einen Prospekt die Länge der einzelnen Beschreibungen in Form der Zeichenanzahl ermittelt werden. Bei der Gelegenheit soll gleichzeitig ermittelt werden, in welchen Beschreibungen der Name „Excel“ auftaucht, da diese Kurse eventuell umbenannt werden sollen. Dies könnte mit der SELECT-Anweisung in Listing 5.13 (MySQL) geschehen.

Listing 5.13
Analyse der Länge der Kursbeschreibungen

```
SELECT
   LENGTH(Kursbeschreibung) "Textlänge",
   POSITION('Excel' IN Kursbeschreibung) AS "Excel vorhanden",
   Kursbeschreibung
FROM tbKursthema;
```

Abbildung 5.13
Ergebnis der Anweisung aus Listing 5.13

Textlänge	Excel vorhanden	Kursbeschreibung
174	0	Windows bietet die Grundlage für alle modernen und anwe...
169	0	Der Kurs wendet sich an PC-Benutzer, die schon etwas Erf...
148	0	Wie schaffe ich Ordnung auf dem PC? Wie können Progra...
121	0	Einstieg in die Textverabeitung mit Word. Grundlegende Fu...
79	0	Einstieg in das Datenbankmanagmentsystem Access. Tab...
115	41	Einstieg in die Tabellenkalkulation mit Excel. Grundfunktion...
92	0	Vertiefung der Arbeit mit MS Access. Komplexe Abfragen u...
59	0	Entwurf relationaler Datenbanken und Umsetzung in MS A...
58	0	Programmierung von Anwendungen auf der Basis von MS ...
90	0	Vertiefung des Umgangs mit Word. Mehrspaltige Formate, ...
73	28	Vertiefung des Umgangs mit Excel, Funktionen einschließli...

Sie sehen vorn die Zeichenanzahl, die mit der Funktion LENGTH ermittelt wurde, und dann die erste Position des Vorkommens von „Excel“, sofern es vorkommt, sonst „0“.

Übungen

Übungen zu den alphanumerischen Funktionen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Es soll ein neuer numerischer Schlüssel für die Kurse verwendet werden. Dazu soll der ASCII-Code des ersten Zeichens der KID mit einer Zufallszahl multipliziert werden und das Ergebnis auf eine ganze Zahl gerundet werden. (U5.6.3.1)
2. Es soll eine Übersicht über die Kurse mit der Kurskennung, dem Kursthema, der geplanten Dauer sowie Beginn- und Endtermin erstellt werden. Dabei sollen die Datenbankkurse zu Access auf MySQL (oder ein anderes Datenbanksystem Ihrer Wahl) umgestellt werden, die Stundenangabe soll den Zusatz `'Stunden'` erhalten. (Ü5.6.3.2)
3. Für Adresslisten soll eine Umformatierung vorgenommen werden. Die Ausgabe soll in der Form: „Bucz, Susanne29xxxCELLEMarxallee12“ erfolgen. Erstellen Sie eine solche Liste, zunächst ohne das Problem des fehlenden Vornamens zu berücksichtigen. (Ü5.6.3.3)
4. Um neue Kurse vorzubereiten, sollen alle Kurs-IDs umbenannt werden. Die neue Kurs-ID soll aus der bisherigen Ortskennung („CE“ oder „H“) und dem Kursbeginn bestehen. (Ü5.6.3.4)

5.6.4 Datumsorientierte Funktionen

Aktuelle Angaben

Zunächst gibt es Funktionen, um das aktuelle Datum, die aktuelle Uhrzeit oder einen kompletten aktuellen Zeitstempel zu erzeugen. Es wird dabei stets auf den aktuellen Rechner zugegriffen und die entsprechenden Werte werden entnommen. Die Benennungen sind unterschiedlich. Das Listing 5.14 zeigt gängige Beispiele.

```
SELECT CURRENT_TIMESTAMP;

SELECT NOW();

SELECT Jetzt();

SELECT SYSDATE();
```

Listing 5.14
Typische Abfragen der aktuellen Rechnerzeit beziehungsweise des Datums

Als Standard können heute die Angaben mit „CURRENT“ angesehen werden, also `CURRENT_DATE`, `CURRENT_TIME` beziehungsweise `CURRENT_TIMESTAMP`.

Ermittlung von Teilen

Die Ermittlung von Bestandteilen eines Datums oder einer Uhrzeit wie Jahr, Monat, Tag, Stunde, Minute oder Sekunde kann zunehmend mit der Funktion `EXTRACT` geschehen. Dabei wird angegeben, welche Einheit, also Jahr, Monat, Tag, Stunde, Minute, Sekunde und einige andere Angaben, aus welcher Zeitangabe extrahiert werden soll. Beispielsweise wird mit (`DAY FROM geburtsdatum`) der Tag aus dem Feld `geburtsdatum` extrahiert und kann dann weiterverwendet werden. Daneben existieren eine Reihe älterer Funktionen für diesen Zweck, die aus Kompatibilitätsgründen weiter verwendet werden. Die Tabelle 5.15 gibt eine Übersicht über gängige Funktionen.

Tabelle 5.15
Gängige Datums-/Zeitfunktionen

SQL-Funktion	Erläuterung	Beispiel
`CURRENT_DATE() -> d` `CURRENT_DATE() -> t`	Das aktuelle Datum (und teilweise auch die Uhrzeit) werden ermittelt.	`CURRENT_DATE() -> '2008-07-20'`
`CURRENT_TIME() -> u`	Die aktuelle Uhrzeit wird ermittelt.	`CURRENT_TIME() -> '20:27:39'`
`CURRENT_TIMESTAMP() -> t`	Liefert den aktuellen Timestamp	`CURRENT_TIMESTAMP () -> '2008-08-20 20:27:39'`
`EXTRACT (Einheit FROM DATE/TIME/TIMESTAMP) -> n`	Standard, um eine Datums-/Zeiteinheit zu ermitteln. `Einheit` kann sein: `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, teilweise zusätzlich mit `TIMEZONE` beispielsweise als `TIMEZONE_HOUR`. Daneben existieren ältere Funktionen wie `YEAR()`, `MONTH()` und andere.	`EXTRACT (YEAR FROM '2008-08-20 20:27:39') -> '2008'`

Tabelle 5.15 (Forts.) Gängige Datums-/Zeitfunktionen

SQL-Funktion	Erläuterung	Beispiel
Zeitintervalle	Es können Differenzen zwischen Datums- oder Uhrzeitangaben berechnet werden und in bestimmten Einheiten ausgedrückt werden. Diese Funktionen sind unterschiedlich gelöst und können teilweise durch das Rechnen mit den Einzelteilen des Datums ersetzt werden.	`DATEDIFF ('2008-07-20', '2008-06-03') -> 47` `TIMEDIFF('20:08:40', '18:07:42') -> '02:00:58'`
Funktionen, um Zeitintervalle zu addieren oder zu subtrahieren	Auch diese Funktionen sind datenbankabhängig.	`'2008-07-20 06:50:04' - TO_DSINTERVAL ('2 08:00:00') -> '2008-07-18 10:50:04' (datenbankabhängig)`

Tabelle 5.16 Datumsorientierte Funktionen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
CURRENT_DATE()	CURRENT_DATE CURDATE() SYSDATE() NOW() UTC_DATE	Now() Jetzt(Date() Datum())	CURRENT_DATE SYSDATE	CURRENT_DATE	CURRENT_DATE NOW()
CURRENT_TIME	CURRENT_TIME CURTIME() SYSTIME() UTC_TIME	Zeit()		CURRENT_TIME	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP NOW() UTC_ TIMESTAMP	Now() Jetzt()	CURRENT_TIMESTAMP SYS TIMESTAMP SYS_EXTRACT_UTC LOCAL TIMESTAMP	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP NOW()

Tabelle 5.16 (Forts.) Datumsorientierte Funktionen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
EXTRACT(Einheit FROM Datums-/ Zeitan-gabe)	EXTRACT (Einheit FROM Datums-/ Zeitan-gabe) WEEKDAY() DAYNAME() DAY() WEEK() MONTH() YEAR() HOUR() MINUTE() SECOND() DAY_ HOUR() DAY_ MINUTE() DAY_ SECOND()	DatePart (Einheit, Feld) DatTeil() Einheiten sind: yyyy, q, m, d, y, w, ww, h, m, s – jeweils als Text in ''. Jahr() Monat() Monats-name() Tag() Stunde() Minute() Sekunde() Wochen-tag() Wochen-tags-name()	EXTRACT (Einheit FROM Datums-/ Zeitan-gabe)	EXTRACT (Einheit FROM Datums- / Zeitan-gabe)	DAYNAME() DAYOF MONTH() DAYOF WEEK() DAYOF YEAR() YEAR() QUARTER() MONTH() MONTHNAME() WEEK() HOUR() MINUTE() SECOND()
Zeitinter-valle	DATE-DIFF() TIME-DIFF() TIMESTAMP DIFF()	IntDat-Diff (Ein-heit, Datum1, Datum2) DatDiff()	TO_DS INTERVAL TO_YM INTERVAL	%	DATEDIFF (Einheit, Datum1, Datum2) Einheiten sind: yy, mm, dd, hh, mi, ss, ms – jeweils in ', also bei-spielsweise 'dd' für Tage.
Funk-tionen, um Zeitinter-valle zu addieren oder zu subtrahie-ren	ADDDATE (Datum, Anzahl, Einheit) ADDTIME (Uhrzeit, Anzahl, Einheit)	DateAdd (Einheit, Anzahl, Datums-feld) Einheit wie DatePart	Durch +/- Operato-ren mit Zeitinter-vallen ADD_ MONTH()	%	%

Beispiel

Als ein weiteres Beispiel wollen wir eine Geburtstagsliste der Personen in der Tabelle `tbPerson` erstellen. Dazu ist es notwendig, dass wir uns auf den Monat und den Tag konzentrieren und die Liste nach diesen ordnen. Um den Monat und den Tag zu extrahieren, können wir die SQL-Standardfunktion

EXTRACT verwenden. Dann wird die Liste nach dem Monat und dem Tag sortiert und Sie erhalten mit der Anweisung in Listing 5.15 das Ergebnis in Abbildung 5.14.

Listing 5.15 Geburtstagsliste mit EXTRACT

```
SELECT
   Familienname, Vorname,
   EXTRACT(MONTH FROM Geburtsdatum) AS "Monat",
   EXTRACT(DAY FROM Geburtsdatum) AS "Tag"
FROM tbPerson
WHERE Geburtsdatum IS NOT NULL
ORDER BY 3 ASC, 4 ASC;
```

Neben der Funktion EXTRACT existieren in den meisten Datenbanksystemen noch eine ganze Reihe älterer Funktionen wie MONTH() oder DAY(), um einzelne Bestandteile eines Datums oder einer Uhrzeit zu extrahieren, die beispielhaft in Listing 5.16 (MySQL) verwendet werden.

Listing 5.16 Geburtstagsliste mit klassischen Datumsfunktionen

```
SELECT Familienname, Vorname, MONTH(Geburtsdatum) AS "Monat",
       DAY(Geburtsdatum) AS "Tag"
FROM tbPerson
WHERE Geburtsdatum IS NOT NULL
ORDER BY 3 ASC, 4 ASC;
```

Abbildung 5.14 Ergebnis der Geburtstagsliste

Familienname	Vorname	Monat	Tag
Schlachter	Dieter	2	2
Sander	NULL	2	5
Martens	Melanie	2	17
Peredy	Helmut	2	23
Ruppert	Nicola	2	25
Weiss	Peter	3	2
Klötzer	Karl	3	13
Bucz	Susanne	4	6
Meier	Kathrin	5	3
Lisewski	Bernd	6	6
Cromberg	Jörg	6	7
Schmidt	Karl	6	25
Karmann	Thomas	8	4
Magerkurth	Melissa	9	4
Weiss	Karin	10	5
Weiss	Peter	11	7
Schulze	Tanja	11	9
Plate	Ulrich	12	2
Winter	Petra	12	30

Übungen zu den datumsorientierten Funktionen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Bestimmen Sie für alle Personen mit Geburtsdatum den Familiennamen, den Vornamen, das aktuelle Datum sowie jeweils die Monate und die Tage zu beiden Daten. (Ü5.6.4.1)
2. Bestimmen Sie für alle Personen den Familiennamen, den Vornamen, das aktuelle Datum, das Geburtsdatum und ermitteln Sie mithilfe des aktuellen Datums und des Geburtsdatums für alle Personen, deren Geburtsdatum bekannt ist und die noch diesen Monat Geburtstag haben, wie viele Tage es noch bis zu ihrem Geburtstag sind (Hinweis: Sollten in dem aktuellen Monat keine Personen mehr Geburtstag haben, addieren oder subtrahieren Sie eine Zahl vom aktuellen Monat.). (Ü5.6.4.2)
3. Ermitteln Sie für alle Kurse in tbKurs die Kurslänge in Tagen. (Ü5.6.4.3)
4. Erstellen Sie eine Liste neuer Kurse, die alle die KID der bisherigen Kurse mit dem Zusatz „-Neuer Kurs“ haben, die Kurskennung und ein Beginndatum, das jeweils 7 Tage nach dem bisherigen Beginndatum liegt. Sie können zur Kontrolle den bisherigen Kursbeginn und den Wochentag hinzufügen. (Ü5.6.4.4)

5.6.5 Datentypumwandlungsfunktionen (Casting)

CAST

Daten werden in verschiedenen Formaten gespeichert: im Wesentlichen als alphanumerische Zeichenketten (String, Text), als Zahlen (ganzzahlig, Gleitkomma), als Zeitangabe (Uhrzeit, Datum, Timestamp, Intervall) oder als binäres Objekt (BLOB, OLE). Viele Funktionen funktionieren innerhalb dieser Datentypklassen. Für die Kombination, Bearbeitung und gerade für die Ausgabe müssen aber auch Umwandlungen zwischen den Datentypen vorgenommen werden können, man spricht von *Casting*. Die Standardfunktion zur Umwandlung von Datentypen heißt CAST mit der Syntax:

```
CAST (Ausdruck AS Datentyp)
```

Dabei wird ein beliebiger Ausdruck in den angegebenen Datentyp umgewandelt, sofern dies möglich ist.

So kann eine Zahl in einen Text umgewandelt werden:

```
SELECT CAST(Stundensatz AS CHAR) FROM tbDozent;
```

In der Praxis tritt die CAST-Funktion allerdings selten auf. Dies liegt auch daran, dass viele Datentypkonvertierungen implizit, also automatisch entsprechend dem Kontext, durchgeführt werden. In Listing 5.17 sind zwei SQL-Anweisungen angegeben, die in MySQL identische Resultate erzeugen.

```
SELECT CONCAT(CAST(Stundensatz AS CHAR),' EUR') FROM tbDozent;

SELECT CONCAT(Stundensatz,' EUR') FROM tbDozent;
```

Listing 5.17
Casting mit CAST und mit implizitem Casting

Der SQL-Interpreter erkennt, dass die Funktion `CONCAT` als Datentypen Texte benötigt, und wandelt den Stundensatz implizit von einem Zahlwert in einen Text um.

Zum anderen sind aber in den Datenbanken typspezifische Funktionen vorhanden, die zur Umwandlung in den gewünschten Datentyp verwendet werden können und oft noch zusätzliche Optionen bieten. Diese Funktionen haben typischerweise Namen wie der Zieltyp oder ähnliche Namen, beispielsweise `DATE()` oder `TO_DATE()`, während die `CAST`-Funktion nur teilweise umgesetzt ist.

Listing 5.18
Umwandlung mit CAST und DATE (MySQL)

```
SELECT
   Beschaeftigungsbeginn,
   DATEDIFF(CAST('2008-07-01' as DATE),Beschaeftigungsbeginn)
   AS "mit CAST",
   DATEDIFF(DATE('2008-07-01'),Beschaeftigungsbeginn) AS "mit DATE"
FROM tbDozent;
```

Abbildung 5.15
Gleiche Ergebnisse mit CAST und typspezifischer Umwandlung

Beschaeftigung...	mit CAST	mit DATE
2003-07-01	1827	1827
2002-01-01	2373	2373
2005-09-15	1020	1020
2003-08-01	1796	1796
2001-01-01	2738	2738

In beiden Spalten wird das Literal in ein Datum umgewandelt und dann eine Differenz von Tagen bestimmt, um zu ermitteln, seit wie vielen Tagen ein Dozent beschäftigt ist.

CONVERT

Neben der Umwandlung von Datentypen können auch innerhalb eines Datentyps die Darstellungen geändert werden. So können diverse Datums- und Zeitdarstellungen ineinander umgewandelt werden. Aber auch Zahlendarstellungen lassen sich umrechnen, so liefert

```
SELECT CONV(15,10,16);
```

in MySQL die hexadezimale Darstellung (zur Basis 16) der Zahl 15, die in dezimaler Darstellung (zur Basis 10) angegeben ist – also einfach „F“.

Tabelle 5.17 Gängige Umwandlungsfunktionen

SQL-Funktion	Erläuterung	Beispiel
`CAST (wert AS Datentyp)`	Wandelt den Wert in eine Darstellung des angegebenen Datentyps um. Dies ist logisch nicht immer möglich. Welche Datentypen in welchen Datentyp umgewandelt werden können, ist in der jeweiligen Dokumentation der Datenbank beschrieben.	`CAST ('12.0' AS DECIMAL)`
`CONVERT (wert, alter Zeichensatz, neuer Zeichensatz)`	Texte können zwischen verschiedenen Zeichensätzen konvertiert werden, soweit diese die verwendeten Zeichen unterstützen. Die Zeichensatznamen sind unterschiedlich, hier sind Oracle-Zeichensätze verwendet worden.	`SELECT CONVERT ('Müller', 'US7ASCII', 'WE8ISO8859P1') FROM DUAL; -> ' Muller'`
Konvertierungen zwischen Zahlenformaten	Zahlen können dezimal, dual oder beispielsweise hexadezimal dargestellt werden.	`CONV(28,10,16) -> 1C`

Tabelle 5.18 Konvertierungsfunktionen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
`CAST (wert as datentyp)`	`CAST() FORMAT (Wert, Datentyp)`	`STR() CSTR() ZString() Cdate() ZDate()` und ähnlich `Ccur(), Cdbl(), Cint(), CIng(), Csng(), Cvdate()`	`CAST() TO_CHAR TO_DATE TO_ TIMSTAMP` sowie eine ganze Reihe weiterer spezieller Funktionen	`CAST()`	`CAST()`
`CONVERT (wert, alter Zeichensatz, neuer Zeichensatz)`	`CURRENT_ TIME CURTIME() UTC_TIME`	`StrConv()` nur für bestimmte Zeichensätze, nicht vergleichbar	`CONVERT()`	%	`CONVERT()`
Konvertierungen zwischen Zahlenformaten	`CONV (Wert, Startbasis, Zielbasis)`	`Hex() Oct()`	verschiedene Funktionen über BIN- und RAW-Formate	%	

Oracle verwendet durchgängig die Funktionen TO_Zieltyp, also etwa `TO_NUMBER()`, `TO_CHAR()`, `TO_DATE()` für die Typumwandlungen.

5.6.6 Logische und sonstige Funktionen

Es gibt natürlich noch eine ganze Reihe weiterer Funktionen, die in SQL-Anweisungen genutzt werden können. In der Praxis wichtig sind insbesondere

- die Generierung eines Schlüssels,
- die Bedingungen und
- die Informationsabfragen.

Der Primärschlüssel einer Tabelle ist häufig eine Nummer ohne inhaltliche Bedeutung. Diese kann prinzipiell von der Datenbank erzeugt werden, wobei die Datenbank dann auch die Eindeutigkeit innerhalb der Tabelle garantiert. Manche Datenbanken bieten dazu einen „Datentyp“ an, der bereits bei der Anlage der Tabelle verwendet werden kann. Manche bieten eine eigene Funktion an, wobei noch zwischen Eindeutigkeit bezüglich der Tabelle und globaler Eindeutigkeit zu unterscheiden ist. Die Tabelle 5.20 bietet einen Überblick.

Tabelle 5.19 Gängige sonstige Funktionen

SQL-Funktion	Erläuterung	Beispiel
Generierung einer ID (`AUTOWERT`)	Wandelt den Wert in eine Darstellung des angegebenen Datentyps um. Dies ist logisch nicht immer möglich. Welche Datentypen in welchen Datentyp umgewandelt werden können, ist in der jeweiligen Dokumentation der Datenbank beschrieben.	`GEN_ID()`
Bedingung (`IF`)	Texte können zwischen verschiedenen Zeichensätzen konvertiert werden, soweit diese die verwendeten Zeichen unterstützen. Die Zeichensatznamen sind unterschiedlich, hier sind Oracle-Zeichensätze verwendet worden.	`IF (1>2,'ja','nein')`
Systemvariable abfragen	Abfrage aktueller Werte wie des Systemdatums oder des aktuellen Benutzers	`SYSTEM_USER()`

Tabelle 5.20
Sonstige Funktionen

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
Generierung einer ID	`UUID()`	Autowert in der Tabelle	über Tabellenbeschreibung	`GEN_ID()`	über `AUTO_INCREMENT` in der Tabelle
Bedingung (`IF`)	`CASE... WHEN... THEN... END` `IF(Bedingung, Ausdruck, Ausdruck)` `IFNULL(...,...,...)` `NULLIF(...,...,...)`	`IF (Bedingung, Ausdruck1, Ausdruck2),` `Switch (bedingung1,Wert1, Bedingung2, Wert2, ...,...,...,...)`	`CASE... WHEN... THEN .. ;` und vergleichbare Funktionen	`CASE... WHEN... THEN ... END`	`CASE... WHEN... THEN ... END`
Systemvariable abfragen	`DATABASE()` `CURRENT_USER` `SCHEMA()` `USER()` `VERSION()`	`CurrentUser()` `Aktueller-Benutzer()`	`UID` `USER` `USERENV`	`%`	`DATABASE()` `CURRENT_USER()`

Die zweite wichtige Funktionsart ist eine Art Verzweigung, die dem IF aus Programmiersprachen ähnelt und etwa der Mächtigkeit des IF in Excel entspricht.

Listing 5.19
Bedingung in einer SELECT-Anweisung (MySQL)

```
SELECT p.Familienname, p.Vorname,
IF(kb.Zeugnis = 'J',CONCAT('Ja am ',k.Kursende),'Nein') AS "Zeugnis"
FROM tbKursbesuche kb INNER JOIN tbKurs k ON (kb.KID = k.KID)
INNER JOIN tbPerson p ON (kb.KTID = p.PID);
```

Beliebt ist auch die Mehrfachverzweigung mit `CASE WHEN`, die in zwei Versionen existiert. Als einfache Bedingung hat sie die Form:

CASE *Bedingung WHEN Ausdruck1 THEN Wert1 {WHEN Ausdruck THEN Wert} [ELSE Wert] [END];*

Dabei wird die Bedingung nacheinander mit den Ausdrücken verglichen. Sobald ein Vergleich eine Übereinstimmung ergibt, wird der zu dem Ausdruck gehörige Wert zurückgegeben. Trifft kein Ausdruck zu, wird der `ELSE`-Wert gewählt. Fehlt der `ELSE`-Ausdruck, wird zumeist `NULL` geliefert.

Die Alternative ist die reine Bedingung, wobei jeweils eine Bedingung einem Wert zugeordnet wird. Die erste Bedingung, die erfüllt ist, bestimmt den Rückgabewert. Für die `ELSE`-Klausel gilt dasselbe wie für das einfache `CASE WHEN`.

CASE WHEN *Bedingung1 THEN Wert1 {WHEN Bedingung THEN Wert} [ELSE Wert] [END];*

Listing 5.20 Einfache CASE WHEN-Bedingung

```
SELECT p.Familienname, p.Vorname,
CASE kb.Zeugnis WHEN 'J' THEN CONCAT('Ja am ',k.Kursende) WHEN 'N'
THEN 'Nein' END AS "Zeugnis"
FROM tbKursbesuche kb INNER JOIN tbKurs k ON (kb.KID = k.KID)
INNER JOIN tbPerson p ON (kb.KTID = p.PID);
```

Listing 5.21 CASE WHEN mit Einzelbedingungen

```
SELECT p.Familienname, p.Vorname,
CASE WHEN kb.Zeugnis = 'J' THEN CONCAT('Ja am ',k.Kursende) WHEN
kb.Zeugnis = 'N' THEN 'Nein' END AS "Zeugnis"
FROM tbKursbesuche kb INNER JOIN tbKurs k ON (kb.KID = k.KID)
INNER JOIN tbPerson p ON (kb.KTID = p.PID);
```

Sowohl Listing 5.19 als auch Listing 5.20 und Listing 5.21 liefern alle dasselbe Ergebnis (MySQL), wie es in Abbildung 0.16 angegeben ist.

Abbildung 5.16 Ergebnis der bedingten Abfrage

Familienname	Vorname	Zeugnis
Sander	NULL	Ja am 2007-04-27
Karmann	Thomas	Nein
Karmann	Thomas	Ja am 2007-04-27
Klötzer	Karl	Ja am 2007-04-27
Plate	Ulrich	Nein
Schmidt	Karl	Nein
Karmann	Thomas	Nein
Klötzer	Karl	Nein
Martens	Melanie	Nein
Müller	Claudia	Nein
Schulze	Tanja	Nein
Winter	Petra	Nein
Ruppert	Nicola	Nein
Magerkurth	Melissa	Nein
Müller	Claudia	Nein
Martens	Melanie	Nein
Lisewski	Bernd	Nein
Peredy	Helmut	Nein

Beachten Sie, dass Oracle auf das abschließende `END` verzichtet und MS Access mit `SWITCH` eine eigene Syntax verwendet.

Übungen

Übungen zu den sonstigen Funktionen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Für Adresslisten soll eine Umformatierung wie in Ü5.6.3.3 vorgenommen werden. Die Ausgabe soll in der Form „Bucz, Susanne 29xxx CELLE Marxallee12“ erfolgen. Erstellen Sie eine solche Liste jetzt mit Berücksichtigung des Problems des fehlenden Vornamens. (Ü5.6.6.1)

2. Jetzt soll ermittelt werden, welche Person im aktuellen Jahr schon Geburtstag hatte oder hat. Dazu soll das Geburtsdatum, das aktuelle Datum und „Ja“ ausgegeben werden, wenn die Person heute oder früher im Jahr Geburtstag hatte, sonst „Nein“. (Ü5.6.6.2)

5.7 Gruppenorientierte Funktionen (Aggregatfunktionen)

Gruppenorientierte Funktionen – auch Aggregatfunktionen genannt – fassen die Werte eines Datenfeldes (Ausdruckes) über eine Gruppe von Datensätzen zusammen. Dadurch wird ein Wert ermittelt, der repräsentativ für das Feld in der gesamten Datensatzgruppe ist.

Beispiel

Soll die Anzahl der Teilnehmer des Kurses „CE23“ ermittelt werden, so ist die Anzahl der Datensätze zu ermitteln. Dies kann wie in Kapitel 4 gesehen relativ einfach geschehen:

Listing 5.22
Einfache Gruppierung mit Zählung der Datensätze

```
SELECT kb.KID, COUNT(kb.KBID) AS "Anzahl Kursbesucher"
FROM tbKursbesuche kb
GROUP BY kb.KID;
```

Es werden hier die Datensätze nach der `KID` gruppiert und ein Gruppendatensatz für jede Gruppe gebildet.

COUNT

Mit der Funktion `COUNT(kb.KBID)` werden die Datensätze gezählt, die einen Eintrag in dem Feld `kb.KBID` besitzen. Sie kennen bereits die Spezialfunktion `COUNT(*)`, die alle Datensätze zählt. `COUNT(*)` stellt im Rahmen der Aggregatfunktionen einen Sonderfall dar, da sie sich als einzige der gängigen Funktionen auf den gesamten Datensatz (daher `*`) bezieht. Alle anderen Aggregatfunktionen beziehen sich immer auf ein einzelnes Feld. Der Vorteil von `COUNT(*)` liegt in der Behandlung der `NULL`-Werte. Da ein gesamter Datensatz niemals `NULL` sein kann, wird mit `COUNT(*)` die Anzahl der Datensätze unabhängig von irgendwelchen `NULL`-Werten bestimmt.

`COUNT(feldname)` bestimmt im Gegensatz dazu die Anzahl der Datensätze, in denen das Feld `feldname` nicht `NULL` ist, kann also unter Umständen weniger Datensätze liefern.

`COUNT(feldname)` ist wiederum eine Kurzform für `COUNT(ALL feldname)`, das die Anzahl aller Werte liefert, die nicht `NULL` sind, während `COUNT(DISTINCT feldname)`, die Anzahl unterschiedlicher Werte berechnet, die nicht `NULL` sind.

Im Fall des Primärschlüsselattributes, hier `KBID`, sind dann aber `COUNT(*)` und `COUNT(KBID)` und `COUNT(DISTINCT KBID)` gleichwertig, da der Primärschlüssel niemals `NULL` sein darf und in jedem Datensatz unterschiedlich sein muss.

Beispiele COUNT

In der Tabelle 5.21 sind unterschiedliche Varianten einer `COUNT`-Anweisung zusammengestellt. Alle Varianten beziehen sich auf die Tabelle `tbPerson`.

Tabelle 5.21 Drei unterschiedliche Zählweisen in einer Tabelle

SELECT-Anweisung	Ergebnis	Erläuterung
`SELECT Count(*) FROM tbPerson;`	20	Alle Datensätze werden berücksichtigt. Kein Datensatz kann den Wert `NULL` aufweisen.
`SELECT Count(Vorname) FROM tbPerson;`	19	Ein Vorname ist ein `NULL`-Wert. Daher werden nur die übrigen Vornamen berücksichtigt.
`SELECT COUNT(DISTINCT Vorname) FROM tbPerson;`	17	Zusätzlich zu dem `NULL`-Wert treten die Vornamen „Karl“ und „Peter“ jeweils doppelt auf. Daher zählen sie nur einmal.

Neben der Aggregatfunktion `COUNT()` gibt es eine ganze Reihe weiterer Aggregatfunktionen.

Die Syntax ist immer

FUNKTIONSNAME ([ALL|DISTINCT] *ausdruck)*

Der Ausdruck sollte dabei nicht selbst eine Aggregatfunktion sein, sondern ein Datenfeld oder eine einfache Berechnung mit Operatoren und Skalarfunktionen.

Tabelle 5.22 Gängige Aggregatfunktionen

SQL-Funktion	Erläuterung
`COUNT(Ausdruck)->n`	Es wird die Anzahl der Werte in den Datensätzen gezählt. `NULL`-Werte werden nicht gezählt. Oracle erlaubt auch ein `COUNT(ALL a1)`, womit nicht nur die unterschiedlichen Werte gezählt werden. `COUNT(*)` zählt alle Datensätze. `NULL`-Werte in irgendwelchen Feldern verringern die Anzahl nicht.
`SUM(Ausdruck)->n`	Es wird die Summe der Werte eines Datenfeldes ermittelt. Nur auf numerische Werte anwendbar.
`AVG(Ausdruck)->n`	Arithmetisches Mittel der Werte des Feldes in einer Gruppe (Durchschnittswert). Nur auf numerische Werte anwendbar.
`MAX(Ausdruck)->Typ des Ausdrucks`	Größter Wert, der in allen Datensätzen auftritt.
`MIN(Ausdruck)-> Typ des Ausdrucks`	Kleinster Wert, der in allen Datensätzen auftritt.
`FIRST(Ausdruck)-> Typ des Ausdrucks`	Ermittelt den Wert des Feldes im ersten Datensatz einer Gruppe.
`LAST(Ausdruck)-> Typ des Ausdrucks`	Ermittelt den Wert des Feldes im letzten Datensatz einer Gruppe.

Tabelle 5.22 (Forts.) Gängige Aggregatfunktionen

SQL-Funktion	Erläuterung
VAR_POP(Ausdruck)/ VAR_SAMP(Ausdruck) ->n	Varianz der Werte. Während die POP-Version von einer Grundgesamtheit (Population) ausgeht, interpretiert die SAMP-Version die Werte als Stichprobe (Sample) einer größeren Grundgesamtheit.
STDDEV_POP(Ausdruck)/ STDDEV_SAMP(Ausdruck) ->n	Standardabweichung der Werte. Während die POP-Version von einer Grundgesamtheit (Population) ausgeht, interpretiert die SAMP-Version die Werte als Stichprobe (Sample) einer größeren Grundgesamtheit.
COVAR_POP(Ausdruck1, Ausdruck2) / COVAR_SAMP(Ausdruck1, Ausdruck2)-> n	Kovarianz zweier Ausdrücke (Felder) als Zusammenhangsmaß. Während POP von einer Grundgesamtheit (Population) ausgeht, interpretiert SAMP die Werte als Stichprobe (Sample) einer größeren Grundgesamtheit.
CORR(Ausdruck1, Ausdruck2) ->n	Korrelationskoeffizient zweier Ausdrücke (Felder). Es wird der Koeffizient von Pearson für numerische Angaben verwendet. Die Ausdrücke müssen daher numerisch sein.

Tabelle 5.23 Aggregatfunktionen in den Datenbanken

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
COUNT()	COUNT()	COUNT()	COUNT()	COUNT()	COUNT()
SUM()	SUM()	SUM()	SUM()	SUM()	SUM()
AVG()	AVG()	AVG()	AVG()	AVG()	AVG()
MAX()	MAX()	MAX()	MAX()	MAX()	MAX()
MIN()	MIN()	MIN()	MIN()	MIN()	MIN()
FIRST()	%	FIRST()	FIRST()	%	
LAST()	%	LAST()	LAST()	%	
STDDEV_ POP()	%	STDevP()	STDDEV_ POP()	%	STDDEV_ POP()
STDDEV_ SAMP()	STD()	STDev()	STDDEV_ SAMP() STDDEV()	%	STDDEV_ SAMP()
VAR_POP()	%	VARP	VAR_POP	%	VAR_POP()
VAR_ SAMP()	%	VAR	VAR_ SAMP() VARIANCE()	%	VAR_SAMP()
COVAR_POP ()	%	%	COVAR_POP( )	%	%

Tabelle 5.23 (Forts.) Aggregatfunktionen in den Datenbanken

ANSI	MySQL	MS Access	Oracle	Firebird	openBase
COVAR_SAMP()	%	%	COVAR_SAMP()	%	%
CORR()	%	%	CORR (Pearson) CORR_S (Spearman) CORR_K (Kendall)	%	%

Regression

Einige der neuen statistischen Aggregatfunktionen sind bereits in die Tabellen aufgenommen worden. Es sind noch weitere Funktionen insbesondere aus dem Bereich der Regression aufgenommen worden, wie `REGR_SLOPE`, `REGR_INTERCEPT`, `REGR_COUNT`, `REGR_R2`, `REGR_AVGX`, `REGR_AVGY`, `REGR_SXX`, `REGR_SYY`, `REGR_SXY`, die aber bisher nur in Oracle in etwas anderer Syntax verfügbar sind.

Beispiel

Es soll für alle Personen ermittelt werden, wie viele Kurse sie besuchen. Zusätzlich soll berechnet werden, wie hoch die bisher gezahlten Beträge sind und wie hoch diese durchschnittlich pro Kurs sind. Da die Personen von besonderem Interesse sind, die besonders viel bezahlt haben, soll das Ergebnis absteigend nach den gezahlten Beträgen sortiert werden. Dafür muss auf die beiden Tabellen `tbPerson` und `tbKursbesuche` zugegriffen werden. Beide Tabellen werden daher über einen `INNER JOIN` verbunden. Dann soll ein Ergebnis pro Person ermittelt werden. Dies könnte über die `PID` in der Tabelle `tbPerson` oder über die `KTID` in der Tabelle `tbKursbesuche` erfolgen. Da die Gruppierung der Beträge letztlich über die Kursbesuche erfolgt, wird die `KTID` gewählt. Jetzt müssen alle Felder des Ergebnisses, außer dem Gruppierungsfeld selbst, mit einer Aggregatfunktion versehen werden, um je Gruppe einen eindeutigen Wert zu liefern. Das gilt letztlich auch für den Namen der Person, wobei einige Datenbanken wie MySQL dies nicht zwingend erfordern. Hier kann aber jede Funktion gewählt werden, die keine numerische Eingabe erfordert. Die Anzahl der Kurse kann über eine Zählung des Primärschlüsselfeldes, hier also der `KBID`, erfolgen, die eindeutig sein muss und keine `NULL`-Werte enthalten kann. Die anderen Funktionen, `SUM` und `AVG`, sind auf das Feld `GezahlterBetrag` anzuwenden, denn schließlich interessiert dessen Summe und Durchschnitt. Insgesamt ergibt sich die `SELECT`-Anweisung in Listing 5.23.

Listing 5.23 Ermittlung interessanter Zahlen für die Kursbesucher

```
SELECT
  MAX(p.Familienname) AS "Familienname",
  MAX(p.Vorname) AS "Vorname",
  COUNT(kb.KBID) AS "Anzahl Kursbesuche",
  SUM(GezahlterBetrag) AS "Gesamtsumme",
  AVG(GezahlterBetrag) AS "Durchschnitt pro Kurs"
FROM tbPerson p INNER JOIN tbKursbesuche kb ON (p.PID = kb.KTID)
GROUP BY kb.KTID
ORDER BY 4 DESC;
```

Das Ergebnis der Anweisung ist in Abbildung 5.17 zu sehen. Es sind wegen des `INNER JOIN` nur die Personen enthalten, die tatsächlich einen Kurs belegt haben.

Familienname	Vorname	Anzahl Kursbesuche	Gesamtsumme	Durchschnitt pro Kurs
Karmann	Thomas	3	880	293.33333333333
Schulze	Tanja	1	350	350
Ruppert	Nicola	1	350	350
Lisewski	Bernd	1	350	350
Müller	Claudia	2	350	175
Klötzer	Karl	2	330	165
Peredy	Helmut	1	300	300
Winter	Petra	1	300	300
Sander	NULL	1	260	260
Martens	Melanie	2	250	125
Schmidt	Karl	1	120	120
Plate	Ulrich	1	80	80
Magerkurth	Melissa	1	50	50

Abbildung 5.17
Ergebnis der Anweisung aus Listing 5.23

Übungen zur SELECT-Anweisung mit GROUP BY-Klausel

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Ermitteln Sie, wie viele Kursteilnehmer mit Gutschein, bar oder mit Überweisung bezahlen, und sortieren Sie die Zahlweise absteigend nach Häufigkeit. (Ü5.7.1)
2. Ermitteln Sie für das Feld `Stundenzahl` über alle Datensätze zumindest die Anzahl, die Summe der Kursstunden, die kleinste und die größte Stundenzahl sowie die durchschnittliche Stundenzahl. Verwenden Sie sinnvolle Alias. (Ü5.7.2)
3. Gruppieren Sie alle Kurse nach der `KID` und geben Sie die Kurskennung, die `KID`, die Summe der Zahlungen sowie die durchschnittliche Zahlungshöhe und wenn möglich die Standardabweichung der Zahlungen an. (Ü5.7.3)
4. Ermitteln Sie den Zusammenhang (Korrelationskoeffizienten) zwischen Rabatt und dem gezahlten Betrag. (Ü5.7.4, nur Oracle)

6

Datenbankinhalte ändern (INSERT, UPDATE, DELETE)

6.1 Neue Datensätze einfügen (INSERT)

Daten, die abgefragt werden sollen, müssen in der Datenbank vorhanden sein. Neue Datensätze in eine Datenbank einzufügen, ist daher eine elementare Aufgabe jeder Datenbank. SQL sieht dafür eine eigene Anweisung vor, die `INSERT INTO`-Anweisung.

6.1.1 INSERT mit Werten und Funktionen

Beispiel

Es soll zunächst in die Tabelle `tbPerson` ein neuer Datensatz für einen neuen Kursteilnehmer mit dem Nachnamen „Gerhardt“ eingefügt werden. Adresse und Geburtsdatum sind gegeben. Leider ist der Vorname nicht bekannt. Der Datensatz kann dann mit folgender SQL-Anweisung eingefügt werden:

Listing 6.1
Einfügen eines Datensatzes in die Personentabelle

```
INSERT INTO tbPerson(
   PID, Familienname, Vorname, PLZ, Ort, Strasse, Geburtsdatum
)
VALUES(
   '51','Gerhardt',NULL,'29221',DEFAULT,'Zöllnerstraße 12',
   '1955-02-24'
);
```

Dabei werden nach der Angabe der Tabelle `tbPerson` zunächst die Feldnamen der Tabellenfelder aufgezählt, die im Folgenden eingefügt werden sollen. Dabei wird eine Reihenfolge vorgegeben, die nicht der Reihenfolge der Felder in der Tabelle entsprechen muss. Dann werden in einer zweiten Liste die Werte aufgezählt, die diesen Feldern zugeordnet werden sollen. Dabei ist die Reihenfolge einzuhalten, die zuvor durch die Liste der Feldnamen bestimmt worden ist. Also wird die `PID` in dem neuen Datensatz auf den Wert „51“ gesetzt, der `Familienname` auf den Wert „Gerhardt“ und so weiter bis zum `Geburtsdatum`.

Bei der Angabe der Werte sind eine Reihe von Bedingungen einzuhalten:

- Es wird ein neuer eindeutiger und noch nicht vorhandener Wert für den Primärschlüssel angegeben. (Gleiches gilt für andere Felder, die als eindeutig definiert sind.)
- Die Werte haben für die Felder passende Datentypen, also Zahlen für numerische Felder, Texte für alphanumerische Felder, Datumsangaben für Felder mit dem Datentyp `DATE` und entsprechend für die anderen Datentypen.
- Es werden für alle Felder, für die in der Tabellendefinition keine `NULL`-Werte erlaubt sind, tatsächlich auch Werte angegeben. Also müssen alle Felder, die nicht leer sein dürfen, mit Werten belegt werden.

Beachten Sie bitte, dass wenn nur eine dieser Bedingungen verletzt ist, der gesamte Datensatz abgewiesen wird. Es erfolgt dann also kein Eintrag in der Datenbank.

Standardwerte

Die obige SQL-Anweisung hat zwei Besonderheiten. Für den Vornamen erfolgt die Angabe `'NULL'`. Dies bedeutet, dass man den Vornamen nicht kennt. Die Frage ist: Was macht die Datenbank mit dieser Angabe? Wir wollen dies gleich klären.

Zweitens ist für den Ortsnamen der Wert `DEFAULT` angegeben. Dies bedeutet, dass der Standardwert der Datenbank verwendet werden soll. Woher soll die Datenbank diesen Wert kennen? Dieser Standardwert wird bei der Anlage der Tabelle angegeben.

Wir wollen uns im Vorgriff die Definition der Tabelle `tbPerson` ansehen, ohne auf die Details jetzt bereits genau einzugehen (siehe Listing 6.2).

Listing 6.2 Definition der Tabelle tbPerson

```
CREATE TABLE IF NOT EXISTS tbPerson (
   PID int NOT NULL PRIMARY KEY,
   Familienname varchar(50) NOT NULL,
   Vorname varchar(50) NULL,
   PLZ char(5) NULL,
   Strasse varchar(50) NULL,
   Ort varchar(50) NULL DEFAULT 'Celle',
   Geburtsdatum date
);
```

Sie sehen in der vierten Zeile an der Angabe `NULL`, dass `NULL`-Werte für den Vornamen zugelassen sind. Die Datenbank übernimmt daher die Angabe `NULL` als `NULL`-Wert. Der Datensatz wird daher nicht abgewiesen. Es gibt einfach keinen Vornamen.

Für den `Ort` wird mit `DEFAULT 'Celle'` der Wert „Celle“ als Standardwert für das Feld `Ort` festgelegt. Die Angabe von `DEFAULT` in der obigen `INSERT`-Anweisung bewirkt daher, dass in dem neuen Datensatz für das Feld `Ort` der Wert „Celle“ eingesetzt wird. Die Umsetzung geschieht bei der Ausführung der `INSERT`-Anweisung. Es gilt also bei der Angabe von `DEFAULT` immer der Standardwert, der bei der Definition der Tabelle festgelegt worden ist. Ist für ein Feld kein Standardwert angegeben, wird das Feld beim Einfügen des Datensatzes mit `NULL` belegt. Ist `NULL` nicht erlaubt, wird der Datensatz abgewiesen.

Die folgende Anweisung würde abgewiesen werden.

Listing 6.3
Unerlaubte INSERT-Anweisung

```
INSERT INTO tbPerson
   (PID, Familienname, Vorname, PLZ, Ort, Strasse, Geburtsdatum)
VALUES
   ('51',NULL,NULL,'29223',DEFAULT,'Hauptstraße 2',1981);
```

Erstens ist der Familienname `NULL`. Dies ist nicht erlaubt und würde für sich allein schon zum Abweisen des Datensatzes führen. Zweitens ist die Angabe `1981` kein gültiges Datum und würde ebenfalls dazu führen, dass die Datenbank den Datensatz nicht einfügt. Wären beide Fehler nicht vorhanden, könnte dann ein Problem entstehen, wenn der Datensatz aus dem ersten Beispiel bereits eingefügt worden wäre. In diesem Fall würde nämlich als `PID` zum zweiten Mal `'51'` verwendet. Dies ist aber der Primärschlüssel, der für alle Datensätze eindeutig sein muss. Zwei Datensätze mit demselben Wert „51“ als `PID` sind aber nicht eindeutig und daher nicht zulässig.

Die Syntax einer INSERT-Anweisung ist damit zunächst klar.

INSERT-Syntax

INSERT INTO *tabellenname [(feldname1, feldname2, ...)]*

VALUES (*wert1, wert2, ...);*

Beide Listen werden jeweils durch Kommata getrennt. Beachten Sie, dass die Feldnamenliste optional ist, also fehlen darf. Fehlt diese Liste, muss die Werteliste Einträge für alle Felder der Tabelle in der richtigen Reihenfolge – entsprechend der Definition der Tabelle – enthalten. Nur so kann der SQL-Interpreter entscheiden, welcher Wert für welches Feld bestimmt ist. Das Weglassen der Feldnamenliste spart natürlich Schreibarbeit. Trotzdem, nutzen Sie dies nur, wenn Sie ausnahmsweise direkt oder testweise einen Datensatz erfassen wollen und sicher sind, welche Struktur die Tabelle zur Zeit hat, da dieses Vorgehen fehleranfällig ist.

Tipp

Bei maschineller Verarbeitung ist das Weglassen der Feldnamenliste fehleranfällig. Wird die Tabellenstruktur verändert, kann die `INSERT`-Anweisung fehlschlagen und der Datensatz abgewiesen werden, sei es, dass Felder fehlen oder Datentypen verändert sind. Im schlimmsten Fall merken Sie es nicht einmal, weil Felder eingefügt worden sind und `NULL`-Werte an den betroffenen Stellen erlaubt waren. Dann landen die Werte der neu eingefügten Datensätze in vollkommen falschen Feldern. Geben Sie also wann immer möglich die Feldnamenliste an.

Als Werte für die Felder sind nicht nur Zahlen, alphanumerische Angaben oder Datumsangaben erlaubt, sondern auch alle Ausdrücke, die einen Wert des richtigen Datentyps für das Feld berechnen. Dies bedeutet insbesondere, dass neben der direkten Wertangabe auch Funktionen verwendet werden können. In der folgenden SQL-Anweisung in MySQL-Syntax wird die Kurskennung durch eine `CONCAT()`-Funktion zusammengesetzt. Der Kursbeginn wird auf zwei Wochen nach dem aktuellen Datum gesetzt. Das Kursende liegt 4 Tage nach dem Kursbeginn:

Listing 6.4 Einfügen eines Datensatzes mit Funktionen

```
INSERT INTO tbKurs (
   KID,
   Kurskennung,
   KTHID,
   KursdauerStunden,
   Kursbeginn,
   Kursende,
   Zertifikat,
   Gebuehr,
   Ustpflichtig,
   DID)
VALUES(
   896,
   CONCAT('CE24','-','Access'),
   3,
   40,
   ADDDATE(Current_Date,14),
   ADDDATE(Kursbeginn,4),
   'N',
   400.00,
   'J',
   NULL);
```

Die verfügbaren Funktionen entsprechen den in Kapitel 5 angesprochenen Funktionen. Sie sind datenbankabhängig.

6.1.2 INSERT mit Unterabfragen

In Kapitel 9 wird auf sogenannte Unterabfragen eingegangen. Dabei wird der Wert eines Feldes durch eine eigene SELECT-Anweisung ermittelt. Lesen Sie gegebenenfalls dort das Prinzip der Unterabfragen (Sub-SELECT) nach.

Listing 6.5 Einfügen eines Datensatzes mit einem Sub-SELECT

```
INSERT INTO tbKurs_Statistik
   ( KID, sum_fehltage, sum_rabatt, mittel_beitrag, Anzahl )
SELECT tbKursbesuche.KID,
   SUM(tbKursbesuche.Fehltage),
   SUM(tbKursbesuche.Rabatt),
   AVG(tbKursbesuche.GezahlterBetrag),
   COUNT(tbKursbesuche.KBID)
FROM tbKursbesuche
GROUP BY tbKursbesuche.KID
HAVING (tbKursbesuche.KID)='CE23';
```

Die Tabelle ist nicht Bestandteil der Kursdatenbank, sondern dient hier nur als Beispiel. Die Anweisung ermittelt zunächst aus der Tabelle `tbKursbesuche` alle Teilnehmer der verschiedenen Kurse. Dabei wird das Ergebnis nach Kursen gruppiert. Dies geschieht über die Identifikationsnummer der Kurse, die `KID`. Dabei entsteht je Kurs ein Datensatz, der alle Informationen über die Kursbesuche dieses Kurses zusammenfasst. Anschließend wird der Datensatz für den Kurs „CE23“ herausgefiltert, sodass letztlich nur ein einziger Datensatz übrig bleibt. Die Werte dieses Datensatzes werden dann paarweise den Feldern der Feldnamenliste zugeordnet. Dabei ist die Reihenfolge entscheidend.

Dem ersten Feld, `KID`, wird das erste Ergebnis `tbKursbesuche.KID` zugeordnet, also die Identifikationsnummer des Kurses selbst. Dies ist zugleich das Gruppierungsfeld, bei dem alle Kursteilnehmer denselben Wert „CE23“ haben. Für die anderen Felder müssen die Werte der Einzeldatensätze der Teilnehmer zu

einem Wert für die Gruppe aggregiert werden. Für das Feld `tbKursbesuche.Fehltage` soll die Anzahl der Fehltage aller Kursteilnehmer summiert werden. Daher wird die Aggregatfunktion `SUM` verwendet. Dem zweiten Feld, `sum_fehltage`, wird mit `SUM(tbKursbesuche.Fehltage)` die Summe der Fehltage aller Teilnehmer des Kurses „CE23“ zugeordnet. An dritter Stelle steht das Feld `sum_rabatt`. Diesem Feld wird über den an dritter Stelle des `SELECT` stehenden Ausdruck `SUM(tbKursbesuche.Rabatt)` die Summe der Rabatte aller Kursteilnehmer als Wert übertragen. Das vierte Feld, `mittel_beitrag`, wird mit dem Mittelwert der gezahlten Beiträge aller Kursteilnehmer `AVG(tbKursbesuche.GezahlterBetrag)` gefüllt. Schließlich wird noch die `Anzahl` der Kursteilnehmer mit `COUNT(tbKursbesuche.KBID)` aus der Anzahl verschiedener Werte des Primärschlüssels ermittelt. Der Primärschlüssel stellt immer eine gute Grundlage für die Ermittlung der Datensatzanzahl dar, da sich dessen Werte nicht wiederholen dürfen.

INSERT INTO ... SELECT ...;

INSERT INTO *tabellenname [(feld1, feld2, ...)]*

SELECT *Select-Anweisung;*

Mehrere Datensätze einfügen

Mithilfe des `SELECT` als Datenquelle können Sie die Werte neuer Datensätze aus bereits existierenden Tabellen übernehmen oder berechnen. Erlaubt sind dabei alle `SELECT`-Anweisungen, die die benötigte Anzahl der Werte im richtigen Format ohne Primärschlüsselverletzung liefern. Mit anderen Worten, alle Anweisungen, die die Bedingungen eines `INSERT` erfüllen. Interessant ist, dass Sie auf diese Weise mit einer einzigen `INSERT INTO`-Anweisung mehrere Datensätze gleichzeitig einfügen können. Jeder Datensatz, den das `SELECT` liefert, wird nämlich nacheinander mit `INSERT` in die Tabelle eingefügt. Hätten wir im obigen Beispiel auf das `HAVING tbKursbsuche.KID = 'CE23'` verzichtet, wäre für alle Kurse ein Datensatz eingefügt worden. Da die Tabelle `tbKursbesuche` die Teilnehmer von vier verschiedenen Kursen enthält, wären vier Gruppendatensätze gebildet und diese vier Datensätze in die Tabelle `tbKurs_Statistik` eingefügt worden.

Die obige `INSERT INTO`-Anweisung ist übrigens das Ergebnis eines mit MS Access generierten SQL-Befehls. Sie sehen, dass man die `HAVING`-Klausel besser durch eine `WHERE`-Bedingung ersetzt hätte, um bereits vor dem Gruppieren zu filtern und so die Datenmenge einzuschränken. Das folgende SQL-Beispiel hätte also dasselbe Ergebnis und bei größeren Datenmengen einen geringeren Speicher- und Zeitbedarf.

Listing 6.6
Einfügen eines Datensatzes mit einer Unterabfrage (Sub-SELECT)

```
INSERT INTO tbKurs_Statistik
   ( KID, sum_fehltage, sum_rabatt, mittel_beitrag, Anzahl )
SELECT tbKursbesuche.KID,
   SUM(tbKursbesuche.Fehltage),
   SUM(tbKursbesuche.Rabatt),
   AVG(tbKursbesuche.GezahlterBetrag),
   COUNT(tbKursbesuche.KBID)
FROM tbKursbesuche
WHERE tbKursbesuche.KID='CE23'
GROUP BY tbKursbesuche.KID;
```

Tipp

Sie werden immer wieder schöne Beispiele für die Erzeugung neuer Datensätze und sogar ganzer Tabelleninhalte mithilfe des `INSERT INTO ... SELECT ...;` finden. Bedenken Sie jedoch, dass dabei Informationen kopiert werden. Dies beinhaltet immer die Gefahr der Redundanz, also der mehrfachen Ablage einer Information. Die Anweisung sollte daher immer sehr bewusst und mit gutem Grund genutzt werden. Dies kann beispielsweise im Fall der Bereitstellung von Informationen als Backup oder für Auswertungen durch Reporting-Werkzeuge sinnvoll sein. Andererseits ist hier gerade bei großen Datenmengen ein spezielles Tool unter Umständen sinnvoller.

Beispiel Archiv

Eine Möglichkeit, in Datenbanken alte Zustände zu konservieren, ist die Erzeugung von historischen Datensätzen, bevor die aktuellen Datensätze verändert werden. Sollen beispielsweise Informationen über unseren Kurs CE23 archiviert werden, weil der Kurs nach seiner Beendigung irgendwann gelöscht werden soll, können die zu archivierenden Informationen in einer eigenen Tabelle `tbKursarchiv` archiviert werden. In diesem Fall werden alle benötigten – teilweise redundanten – Informationen in jeweils einem Datensatz zusammengefasst.

Listing 6.7
Einfügen von Archivdatensätzen aus mehreren Tabellen

```
INSERT INTO tbKursarchiv (
   KID, Kursthema, Kurskennung, Kursbeginn, Kursende, PID,
   Familienname, Vorname, archiviert
)
SELECT
   tbKurs.KID, tbKursthema.Kursthema, tbKurs.Kurskennung,
   tbKurs.Kursbeginn, tbKurs.Kursende, tbPerson.PID,
   tbPerson.Familienname, tbPerson.Vorname,
   current_date() AS archiviert
FROM tbPerson
   INNER JOIN ((tbKursthema INNER JOIN tbKurs
                        ON (tbKursthema.KTHID = tbKurs.KTHID))
   INNER JOIN tbKursbesuche ON (tbKurs.KID = tbKursbesuche.KID))
                        ON (tbPerson.PID = tbKursbesuche.KTID)
WHERE (tbKurs.KID='CE23');
```

Die Ergebnisdatensätze sind in Abbildung 6.1 zu sehen. Diese Datensätze werden durch die `INSERT`-Anweisung in der Archivtabelle abgelegt. Beachten Sie bitte, wie die Reihenfolge der Feldnamen in der Zieltabelle wiederum genau mit der Reihenfolge in der `SELECT`-Anweisung übereinstimmt. Dies entspricht genau der Übereinstimmung, die auch bei der Angabe der Feldnamen und der zuzuordnenden Werte in der Grundform des `INSERT INTO` wie wir sie am Anfang hatten, eingesetzt wurde. Wieder muss bei der `INSERT INTO`-Anweisung die Liste der Zielfelder stets der Reihenfolge der Werte – egal wie sie ermittelt werden – entsprechen.

Abbildung 6.1
Ergebnis der Archivierung mithilfe einer INSERT INTO-Anweisung

KID	Kursthema	Kurskennung	Kursbeginn	Kursende	PID	Familienname	Vorname	archiviert
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	4	Karmann	Thomas	18.08.2007 22:12:21
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	5	Klötzer	Karl	18.08.2007 22:12:21
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	15	Martens	Melanie	18.08.2007 22:12:21
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	10	Müller	Claudia	18.08.2007 22:12:21
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	31	Schulze	Tanja	18.08.2007 22:12:21
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	32	Winter	Petra	18.08.2007 22:12:21
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	24	Ruppert	Nicola	18.08.2007 22:12:21
CE23	Access I	Celle23-Access	06.08.2007	10.08.2007	37	Magerkurth	Melissa	18.08.2007 22:12:21

6.1.3 INSERT mit SET

SET-Syntax

Eine Alternative zu der INSERT-Anweisung mit Feldnamenliste und Werteliste ist die SET-Syntax. Dabei werden die einzelnen Felder namentlich erwähnt und ihnen werden wie sonst auch Werte über Ausdrücke zugewiesen. Allerdings erfolgt die Zuordnung von Feldname und Wert nicht über die Reihenfolge in einer Liste, sondern über eine ausdrückliche Zuordnung von Feldname und Wert.

Listing 6.8
Einfügen eines Datensatzes mit der SET-Syntax

```
INSERT INTO tbKurs SET
   KID=894,
   Kurskennung=CONCAT('CE24','-','Access'),
   KTHID=3,
   KursdauerStunden=40,
   Kursbeginn=ADDDATE(CURRENT_DATE,14),
   Kursende=ADDDATE(Kursbeginn,4),
   Zertifikat='N',
   Gebuehr=400.00,
   Ustpflichtig='J',
   DID=NULL;
```

Diese Schreibweise ähnelt der Syntax, wie wir sie bei der Änderung einzelner Werte in Datensätzen (UPDATE) sehen werden. Alle in einem SET nicht erwähnten Felder werden mit den Standardwerten (DEFAULT) belegt. Alle Felder, die nicht erwähnt sind und für die auch kein DEFAULT-Wert vorliegt, werden – Sie ahnen es schon – mit NULL belegt. Sind Felder in der Tabellendefinition mit NOT NULL definiert, sind NULL-Werte also nicht erlaubt, wird die INSERT-Anweisung abgewiesen und der komplette Datensatz wird nicht geschrieben.

6.1.4 Besonderheiten des INSERT mit MS Access

MS Access

Die Oberfläche von MS Access verfügt über spezielle Mechanismen zum Einfügen, Ändern und Löschen von Datensätzen. Die Aktivierung der entsprechenden Mechanismen ist einfach. Sie gehen dazu zunächst in die Entwurfsansicht einer „normalen“ Abfrage. Hier finden Sie den Menübefehl ABFRAGE. Nach Auswahl dieses Befehls finden Sie eine Übersicht über die zur Verfügung stehenden Abfragetypen (siehe Abbildung 6.2).

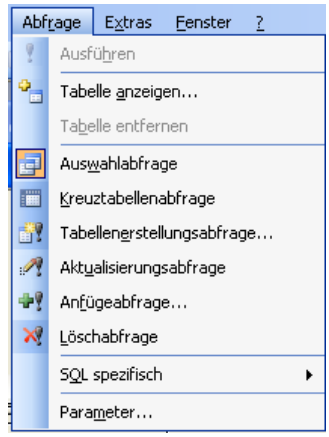
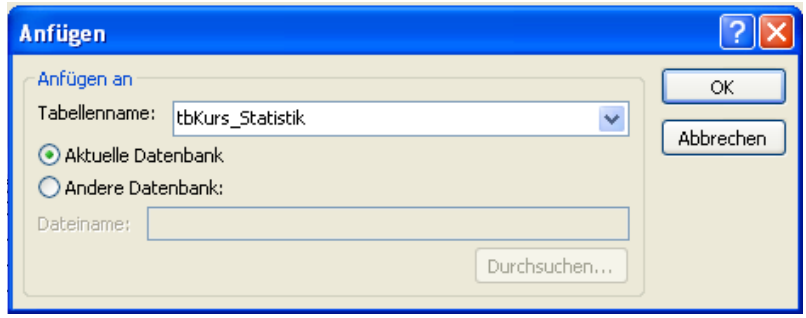


Abbildung 6.2
Spezialabfragen in MS Access

Abbildung 6.3
Zieltabelle einer Anfügeabfrage (INSERT INTO)



Damit haben Sie die Zieltabelle des `INSERT INTO` festgelegt. Die Tabelle `tbKurs_Statistik` beinhaltet für jeden Kurs ein Feld `KID` für die Identifikationsnummer, ein Feld `sum_fehltage` für die Anzahl der Fehltage aller Kursteilnehmer, ein Feld `sum_rabatt` für die Summe der Rabatte, ein Feld `mittel_beitrag` für den Mittelwert der Beiträge und ein Feld `Anzahl` für die Anzahl der Kursteilnehmer.

Wir müssen die Abfrage jetzt so konstruieren, dass alle Felder dieser Zieltabelle mit sinnvollen Werten belegt werden können. Als Quelle dieser Werte kommen feste Werte, Funktionen oder andere Tabellen aus der Datenbank infrage. Wir wollen hier beispielhaft die Tabelle `tbKursbesuche` verwenden, der wir alle Werte entnehmen können. Dazu wird die Tabelle wie bei Auswahlabfragen üblich, zunächst in die Abfrage übernommen. Wichtig ist also, dass oben in der Abfrage die Quelle der Informationen steht, das Ziel ist bereits festgelegt worden.

Nachdem Sie die Abfrage als Anfügeabfrage markiert haben, wird im unteren Bereich des Abfragefensters zusätzlich eine neue Zeile zur Verfügung gestellt. In der neuen Zeile ANFÜGEN AN können Sie die Namen der Felder in der Tabelle `tbKurs_Statistik` angeben, die Sie gerade als Zieltabelle gewählt haben (siehe Abbildung 6.4). Diese Feldnamen stellen das Ziel der `INSERT`-Anweisung dar, hier werden die Informationen gespeichert. Die Zeile FELD beinhaltet die Quelle der Werte, also die Werte und Ausdrücke, die in die entsprechenden Felder eingetragen werden sollen. Die Werte in der oberen Zeile werden also in die Felder der unteren Zeile ANFÜGEN AN geschrieben. Die Zuordnung erfolgt paarweise, oben der Wert, unten der Feldname.

Sie können die einzelnen Felder durch Aufklappen auswählen. Jetzt müssen Sie noch festlegen, wie die Werte berechnet werden sollen. Dazu nutzen Sie die bereits bekannte Vorgehensweise für Auswahlabfragen. In diesem Fall müssen Sie noch die Gruppierungsfunktion einschalten und die einzelnen Aggregatfunktionen *Summe*, *Mittelwert* und *Anzahl* für die jeweiligen Felder auswählen. Diese Beschreibung ist bekannt. Der einzige Unterschied zu Auswahlabfragen besteht darin, dass das Ergebnis nicht am Bildschirm dargestellt, sondern direkt mit `INSERT` in die Zieltabelle geschrieben wird.

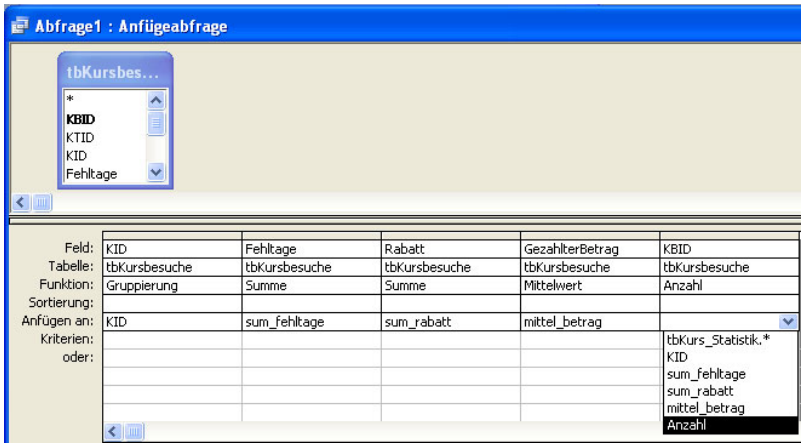


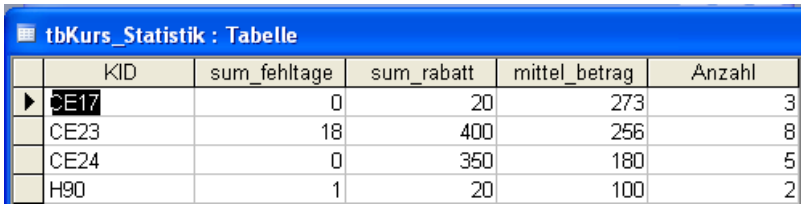
Abbildung 6.4
MS Access-Anfügeabfrage

Aus diesen Angaben erzeugt MS Access eine INSERT-Anweisung. Dies ist – mit Alias – genau die SQL-Anweisung, die wir bereits betrachtet haben, nur ohne WHERE- beziehungsweise HAVING-Klausel.

Listing 6.9
INSERT aus MS Access

```
INSERT INTO tbKurs_Statistik
   ( KID, sum_fehltage, sum_rabatt, mittel_beitrag, Anzahl )
SELECT
   tbKursbesuche.KID,
   Sum(tbKursbesuche.Fehltage) AS [Summe von Fehltage],
   Sum(tbKursbesuche.Rabatt) AS [Summe von Rabatt],
   Avg(tbKursbesuche.GezahlterBetrag) AS [Mittelwert von
       GezahlterBetrag],
   Count(tbKursbesuche.KBID) AS [Anzahl von KBID]
FROM tbKursbesuche
GROUP BY tbKursbesuche.KID;
```

Sie können die Abfrage jetzt testen, indem Sie über das Symbol links oben in die Datenblattansicht umschalten, und erhalten die vier Datensätze, die eingefügt wurden (Abbildung 6.5).



KID	sum_fehltage	sum_rabatt	mittel_betrag	Anzahl
CE17	0	20	273	3
CE23	18	400	256	8
CE24	0	350	180	5
H90	1	20	100	2

Abbildung 6.5
Ergebnis des Tests der Anfügeabfrage

Sie sehen die automatisch erzeugten Alias als Überschriften der Ergebnisspalten. Mit ABFRAGE/AUSFÜHREN oder dem nebenstehend abgebildeten Symbol, können Sie die Anfügeabfrage wirklich ausführen.

Jetzt wird die INSERT INTO-Anweisung ausgeführt. Da dies zu einer realen und nicht automatisch rückgängig zu machenden Änderung der Datenbank führt, müssen Sie die Anweisung bestätigen.

Nach der Bestätigung können Sie das Ergebnis in der Tabelle `tbKurs_Statistik` betrachten. Hier sollten genau die vier Datensätze aus der Abbildung 6.5 vorhanden sein. Um das obige Beispiel zu vervollständigen, soll jetzt gezielt nur der Kurs „CE23“ in die Statistiktabelle eingefügt werden. Löschen Sie dazu zunächst noch einmal die schon vorhandenen Datensätze aus der Tabelle `tbKurs_Statistik`. Ändern Sie dann Ihre Anfügeabfrage so, dass Sie einen Filter auf den Kurs „CE23“ setzen (Abbildung 6.6), und führen Sie die Abfrage noch einmal aus. Sie sollten jetzt noch gefragt werden, ob Sie einen Datensatz einfügen wollen. Nach der Bestätigung sollte dieser Datensatz mit der `KID` „CE23“ wieder in der Tabelle `tbKurs_Statistik` enthalten sein.

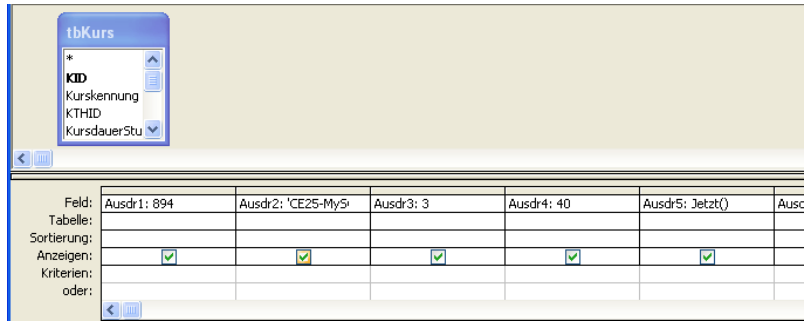
Abbildung 6.6
Anfügeabfrage mit Filter

Feld:	KID	Fehltage	Rabatt	GezahlterBetrag	KBID
Tabelle:	tbKursbesuche	tbKursbesuche	tbKursbesuche	tbKursbesuche	tbKursbesuche
Funktion:	Gruppierung	Summe	Summe	Mittelwert	Anzahl
Sortierung:					
Anfügen an:	KID	sum_fehltage	sum_rabatt	mittel_betrag	Anzahl
Kriterien:	='CE23'				
oder:					

Entsprechend können Sie alle anderen Funktionen, die Sie von Auswahlabfragen kennen, hier nutzen, um neue Datensätze einzufügen. Wichtig ist nur, dass die Werte zu den Feldern der Zieltabelle passen.

Als Werte können natürlich nicht nur Daten aus anderen Tabellen, sondern sowohl Literale, wie `'CE23'` als `KID` oder `'12'` als `sum_fehltage`, als auch Funktionen wie `Jetzt()` verwendet werden. In SQL werden bei MS Access daraus immer eine Feldnamenliste und eine Unterabfrage (`SELECT`), die die Werte den Feldern paarweise zuordnet. Als weiteres Beispiel wollen wir uns dazu noch eine Einfügeabfrage für die Tabelle `tbKurs` ansehen (Abbildung 6.7).

Abbildung 6.7
Einfügeabfrage mit festen Werten und einer Funktion



Daraus wird in SQL:

Listing 6.10
Einfügeabfrage mit Literalen und Funktion

```
INSERT INTO tbKurs (
   KID, Kurskennung, KTHID, KursdauerStunden, Kursbeginn, Kursende,
   Zertifikat, Gebuehr, Ustpflichtig, DID )
SELECT 894 AS Ausdr1, 'CE25-MySQL' AS Ausdr2, 3 AS Ausdr3,
       40 AS Ausdr4, Now() AS Ausdr5, Ausdr5+18 AS Ausdr6,
       0 AS Ausdr7, 400 AS Ausdr8, -1 AS Ausdr9, 812 AS Ausdr10;
```

Sie sehen, dass MS Access wiederum eine Anweisung mit SELECT erzeugt hat, obwohl in der Abfrage auf keine andere Tabelle Bezug genommen wurde. Daher besteht das SELECT aus der einfachen Form ohne FROM-Klausel.

Die Angaben Ausdr1, Ausdr2 und die weiteren Angaben werden von MS Access wieder automatisch als Alias generiert. Sie können sie verwenden, um sich in Ihren Wertbestimmungen auf andere Ausdrücke zu beziehen, wie Sie am Ausdruck

```
Ausdr5+18 AS Ausdr6
```

für das Kursende erkennen können. Natürlich können Sie die Alias umbenennen und eigene Ausdrücke verwenden.

Wenn Sie diese Abfrage in MS Access ausführen, fragt MS Access Sie wiederum in einer Kontrollabfrage, ob Sie den neuen Datensatz wirklich einfügen wollen. Sie haben also über die Oberfläche die INSERT-Anweisung richtig erzeugt und können Sie ausführen.

Die Abfrage mit dem SELECT als Datenquelle sieht relativ kompliziert aus. Viel einfacher könnte MS Access auch mit einer Feldnamenliste und einer Werteliste arbeiten und dann etwa folgende INSERT-Anweisung generieren:

Listing 6.11
Alternative vereinfachte Einfügeabfrage

```
INSERT INTO tbKurs (
   KID, Kurskennung, KTHID, KursdauerStunden, Kursbeginn, Kursende,
   Zertifikat, Gebuehr, Ustpflichtig, DID )
VALUES (894,'CE25-MySQL',3,40,Now() AS a, a+18,0,400,-1,812);
```

Diese Anweisung erfüllt denselben Zweck und würde normalerweise immer verwendet werden, wenn die Anweisung manuell eingegeben wird. Für eine automatisch generierte Anweisung stellt der zusätzliche „Schreibaufwand“ aber kein Hindernis dar. Ein weiterer Vorteil des SELECT ist die größere Allgemeingültigkeit der SELECT-Anweisung als Datenquelle gegenüber der VALUES-Syntax.

Tipp

Wenn Sie Anfügeabfragen in MS Access erstellen, ist die gesamte Beschreibung der Datenherkunft mit einer normalen Auswahlabfrage identisch. Sie können das Ergebnis in Ruhe testen, indem Sie in die Datenblattansicht wechseln (Symbol links in Symbolleiste). Was Sie hier als Ergebnis der Abfrage sehen, wird genau das sein, was die Datenbank bei einer Anfügeabfrage an das INSERT INTO als Daten schickt. Sie ändern aber noch nichts in der Datenbank.

Haben Sie das gewünschte Ergebnis zusammengestellt, führen Sie die Anfügeabfrage tatsächlich aus (ABFRAGE/AUSFÜHREN).

6.1.5 INSERT-Übungen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine INSERT-Anweisung.

1. Fügen Sie ein neues Kursthema in die Tabelle tbKursthema ein. Das neue Thema soll die KTHID „12“ haben, „MySQL“ heißen und 40 Stunden umfassen. Voraussetzung ist der Kurs Nummer 2. Die Kursbeschreibung bleibt Ihnen überlassen, die Kursdemo ist unbekannt. (Ü6.1.1)

2. Richten Sie einen neuen Kurs „CE25“ zu dem eben angegebenen Thema ein. Der Kurs soll am Montag, dem 27.8.2009 beginnen und bis Freitag gehen. Der Kurs soll 40 Stunden umfassen und mit einem eigenen Zertifikat abschließen. Als Kursleiter soll Peter Weiss aus Hannover eingesetzt werden. Der Kurs ist umsatzsteuerpflichtig und die Gebühr beträgt 400,- €. (Ü6.1.2)
3. Zu dem eben eingerichteten Kurs hat sich Ulrich Plate angemeldet. Er erhält die KBID „443“. Er bezahlt selbst per Überweisung und hat noch nichts eingezahlt. Rabatt und Fehltage sind die Standardwerte. Ein Zeugnis wird nicht erstellt. (Ü6.1.3)
4. Es hat sich noch ein neuer Teilnehmer angemeldet, Frau Martina Kasten aus 30514 Hannover, Am Sägewerk 12. Das Geburtsdatum ist unbekannt. Erstellen Sie alle notwendigen Einträge. Sie hat sich zu denselben Konditionen wie Ulrich Plate angemeldet. (Ü6.1.4)
5. Was bewirkt die folgende SQL-Anweisung theoretisch? Welche Fehler werden wahrscheinlich bei der Ausführung auftreten?

```
INSERT INTO tbKurs (
KID, Kurskennung, KTHID, KursdauerStunden, Kursbeginn,
  Kursende, Zertifikat, Gebuehr, Ustpflichtig, DID )

SELECT KID, Kurskennung, KTHID, KursdauerStunden,
       Kursbeginn+7, Kursende+7, DEFAULT, Gebuehr, Ustpflichtig,
       NULL
FROM tbKurs;
```

6. Zum 1.1.2009 ist die Person mit der PID 34 zum Dozententeam gestoßen. Er arbeitet selbstständig zum Stundensatz von 15 €. Titel und Qualifikation sind nicht bekannt. Fügen Sie ihn mit der DID 835 ein.

Wenn Sie jetzt den Ausgangszustand wiederherstellen wollen, sollten Sie in Abschnitt 6.3.4 die erste Übung durchführen. Lesen Sie dort auch die vorausgehenden Hinweise zum DELETE-Befehl. Sie können aber auch hier weiterarbeiten und die Löschungen vornehmen, wenn Sie in dem Abschnitt angekommen sind.

6.2 Vorhandene Datensätze ändern (UPDATE)

Datensätze können natürlich nicht nur eingefügt werden, sondern vorhandene Datensätze können auch geändert werden. Stellen Sie sich vor, die Preise von Produkten sollen geändert werden, Menschen wechseln die Wohnung, ziehen um oder heiraten und ändern ihren Namen. In Kursen sind die Zahlungen zu ändern, Fehltage zu erhöhen und Zeugnisse auszustellen. Es gibt also viele gute Gründe, Datensätze in der Datenbank zu ändern.

Daher bietet SQL auch eine Anweisung, um Datensätze zu ändern. Diese Anweisung heißt UPDATE.

6.2.1 UPDATE-Anweisungen

Beispiel

Das neue Kursprogramm soll im Kursthema bereits darauf hinweisen, dass es sich um EDV-Kurse handelt, da künftig auch andere Kursarten geplant sind. Um die Datenbankstruktur nicht ändern zu müssen, soll daher allen Kursthemen ein „EDV: “ vorangestellt werden. Dies kann mit einem

```
UPDATE tbKursthema SET
kursthema = concat('EDV: ',kursthema);
```

geschehen. Sie sehen bereits an der Syntax, dass die Tabelle `tbKursthema` geändert werden soll. Dabei wird die bereits von der `INSERT`-Anweisung bekannte Syntax mit `SET` verwendet. Mit einer `UPDATE`-Anweisung können ein oder mehrere Felder gleichzeitig geändert werden. Es können auch mehrere und sogar alle Datensätze einer Tabelle mit einer einzigen `UPDATE`-Anweisung geändert werden.

Rückgängig machen

Eine `UPDATE`-Anweisung verändert Ihre Datenbank. Dies ist – sofern Sie nicht wie in Kapitel 13 beschrieben Transaktionen verwenden – unwiderruflich. Das aus Windows gewohnte „Rückgängig“ gibt es in SQL nicht. Bevor Sie eine `UPDATE`-Anweisung daher auch nur testweise in Ihrer Datenbank verwenden, sollten Sie sich angewöhnen, das Ergebnis zuvor so gut wie möglich zu testen. Sie sollten daher immer zunächst eine „normale“ `SELECT`-Anweisung verwenden, um zu testen, welche Datensätze Sie ändern würden. Die `SELECT`- und die `UPDATE`-Anweisung arbeiten hinsichtlich der betroffenen Datensätze identisch. Bei den mit `SELECT` ermittelten Datensätzen sollten Sie zumindest den Primärschlüssel, die bisherigen Werte der zu ändernden Felder und die neuen Werte anzeigen.

```
SELECT KTHID, kursthema, concat('EDV: ',kursthema)
FROM tbKursthema;
```

Dies ergibt die Datensätze aus der Abbildung 6.8.

KTHID	kursthema	concat('EDV: ',kursthema)
1	Einstieg mit Windows	EDV: Einstieg mit Windows
2	Windows für Fortgeschrittene	EDV: Windows für Fortgeschrittene
3	Ordnung im Dateisystem	EDV: Ordnung im Dateisystem
4	Word I	EDV: Word I
5	Access I	EDV: Access I
6	Excel I	EDV: Excel I
7	Access II	EDV: Access II
8	Datenbankentwurf	EDV: Datenbankentwurf
9	VBA-Datenbankprogrammierung	EDV: VBA-Datenbankprogrammierung
10	Word II	EDV: Word II
11	Excel II	EDV: Excel II

Abbildung 6.8
Ergebnis des SELECT in Vorbereitung auf die UPDATE-Anweisung

Nachdem Sie mit dem Ergebnis zufrieden sind, können Sie dann die entsprechende `UPDATE`-Anweisung wie oben beschrieben formulieren und ausführen.

Bei jeder `UPDATE`-Anweisung muss der SQL-Interpreter wissen, welche Spalten und welche Zeilen der Tabelle geändert werden sollen. Die Spalten ergeben sich aus den Feldnamen, die in der `SET`-Anweisung aufgeführt werden. Machen Sie keine weiteren Angaben, bezieht sich die Änderung auf die gesamte Tabelle, was in den wenigsten Fällen gewünscht ist.

Achtung

Eine UPDATE-Anweisung ohne WHERE-Klausel ändert alle Datensätze in der Tabelle. Das ist meistens nicht erwünscht. Daher muss jede UPDATE-Anweisung vor ihrer Ausführung immer sorgfältig getestet werden.

Einschränken lassen sich die Zeilen durch die bereits bekannte WHERE-Klausel, womit sich dann die Syntax der UPDATE-Anweisung ergibt.

UPDATE-Syntax

UPDATE *tabellenname*

SET *feldname1=ausdruck1 [{, feldname = ausdruck}]*

[WHERE *bedingungsliste];*

Beispiel

Jetzt sollen alle Kurse, die sich auf „Access“ beziehen, den Zusatz „Datenbank: “ erhalten. Außerdem soll die Kursdauer dieser Kurse um 50 % erhöht werden.

Zunächst testen wir mit einer SELECT-Anweisung.

Listing 6.12
Testweise Auswahl aller Kurse zum Thema Access (MySQL)

```
SELECT
   KTHID,
   Kursthema,
   CONCAT('Datenbank: ',SUBSTRING(Kursthema,6)),
   DauerPlan,
   ROUND(DauerPlan * 1.5,0)
FROM tbKursthema
WHERE Kursthema LIKE '%Access%';
```

Neu ist in der Anweisung die WHERE-Klausel, die dafür sorgt, dass UPDATE auf die Datensätze beschränkt wird, die „Access“ in der Kursbeschreibung enthalten. Im Ergebnis erhalten wir zwei Zeilen mit den gewünschten Ergebnissen. Danach wird diese SELECT-Anweisung mit wenigen Änderungen in eine UPDATE-Anweisung umgewandelt.

Listing 6.13
Die Anweisung als UPDATE-Anweisung (MySQL)

```
UPDATE tbKursthema SET
Kursthema = CONCAT('Datenbank: ',SUBSTRING(Kursthema,6)),
DauerPlan = ROUND(DauerPlan * 1.5,0)
WHERE Kursthema LIKE '*Access*';
```

Das Ergebnis ist eine geänderte Tabelle tbKursthema, in der die beiden Datenbankkurse jetzt einen entsprechenden Zusatz erhalten haben, wie in Abbildung 6.9 zu sehen.

Abbildung 6.9
Gezieltes Update zweier Datensätze mit WHERE-Klausel

KTHID	Kursthema	Kursbeschreibung	Kursdemo	DauerPlan	Voraussetzung
1	EDV: Einstieg mit Windows	Windows bietet die Grundlage für alle mode...		20	NULL
2	EDV: Windows für Fortge...	Der Kurs wendet sich an PC-Benutzer, die s...		20	1
3	EDV: Ordnung im Dateisy...	Wie schaffe ich Ordnung auf dem PC? Wie...		8	2
4	EDV: Word I	Einstieg in die Textverabeitung mit Word. G...		40	2
5	Datenbank: Access I	Einstieg in das Datenbankmanagmentsyste...		60	2
6	EDV: Excel I	Einstieg in die Tabellenkalkulation mit Excel...		40	2
7	Datenbank: Access II	Vertiefung der Arbeit mit MS Access. Kompl...		60	5
8	EDV: Datenbankentwurf	Entwurf relationaler Datenbanken und Ums...		80	7
9	EDV: VBA-Datenbankpro...	Programmierung von Anwendungen auf der...		80	7
10	EDV: Word II	Vertiefung des Umgangs mit Word. Mehrsp...		40	4
11	EDV: Excel II	Vertiefung des Umgangs mit Excel, Funktio...		40	6

Die UPDATE-Anweisung besitzt eine Reihe von Erweiterungen. So kann mit LOW PRIORITY die Ausführung des UPDATE bei hoher Belastung der Datenbank verzögert werden. Ein UPDATE erfordert Sperren in der Datenbank, die dann andere Benutzer unter Umständen behindern. Ist die unmittelbare Ausführung nicht notwendig, kann dies daher sinnvoll sein. Mehr dazu in Kapitel 13. Zu Testzwecken kann auch die Anzahl der zu ändernden Datensätze beschränkt werden.

UPDATE [LOW PRIORITY] *tabellenname*

SET *feldname=ausdruck1 [{, feldname = ausdruck}]*

[WHERE *bedingungsliste]*

[LIMIT *anzahl];*

Weitere Zusätze sind für die einzelnen Datenbanken verfügbar.

6.2.2 Besonderheiten von UPDATE bei MS Access

MS Access bietet auch für die Erstellung von UPDATE-Anweisungen eine eigene Möglichkeit im Rahmen der grafischen Benutzeroberfläche. Dazu wird wiederum zunächst eine neue Abfrage geöffnet und im Entwurfsmodus über ABFRAGE/AKTUALISIERUNGSABFRAGE auf die sogenannte Aktualisierungsabfrage umgeschaltet.

Wie gewohnt kann jetzt die Datenquelle für die Abfrage gewählt werden. Neu ist allerdings, dass Datenquellen zugleich das Ziel der Änderung, also des Updates, bilden können. Dazu wird wiederum im unteren Teil des Entwurfsfensters die neue Zeile AKTUALISIEREN eingeblendet.

Jetzt ist jede Spalte so zu lesen, dass der Feldname in der Zeile FELD das Ziel der Aktualisierung angibt, während der in derselben Spalte unter AKTUALISIEREN stehende Ausdruck den Wert bestimmt, der in dem Feld gespeichert werden soll. Die Abbildung 6.10 zeigt eine solche Situation, bei der das Feld Kursthema aktualisiert wird. Es können weitere Felder in die Abfrage aufgenommen werden, die als Filter oder für andere Zwecke dienen. Auch zu aktualisierende Felder können ein weiteres Mal aufgenommen werden.

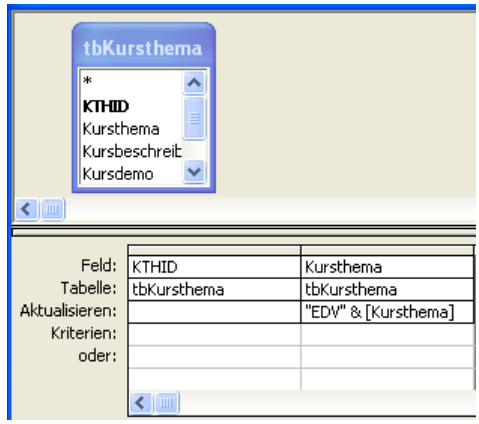


Abbildung 6.10
Aktualisieren des Kursthemas

MS Access generiert aus diesen Angaben eine SQL-Anweisung, die wie in Listing 6.14 aussieht.

Listing 6.14 UPDATE-Anweisung in MS Access

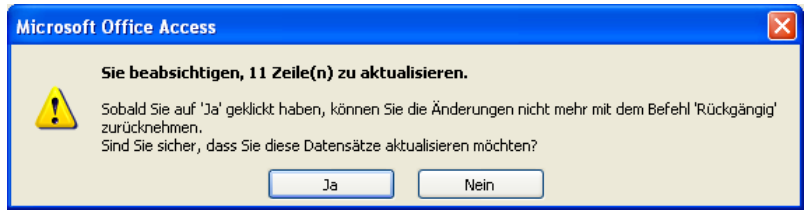
```
UPDATE tbKursthema
SET tbKursthema.Kursthema = "EDV:" & [Kursthema];
```

Beachten Sie die Syntax: Das &-Zeichen repräsentiert hier die Funktion CONCAT, wie sie andere Datenbanksysteme verwenden würden. Sie können diese Anweisung testen, indem Sie mit dem nebenstehenden Icon auf die Datenansicht umschalten. Allerdings können Sie damit nur die ausgewählten Datensätze überprüfen, nicht aber die Funktionsfähigkeit der Ausdrücke für die Aktualisierung. Sind Sie hier unsicher, müssen Sie zunächst eine Auswahlabfrage erstellen und die Ausdrücke in der obersten Zeile FELD als berechnete Felder testen. Sie können dort die Ausdrücke identisch verwenden. Dies ist zwar etwas umständlich, schützt aber vor unliebsamen Überraschungen.

Sobald Sie die Aktualisierungsabfrage tatsächlich ausführen – entweder mit ABFRAGE/AUSFÜHREN oder mit dem entsprechenden nebenstehenden Symbol –, wird die Änderung tatsächlich vorgenommen. Zur Sicherheit erscheint noch eine Kontrollabfrage (Abbildung 6.11), die noch einmal zumindest eine Plausibilitätsprüfung der Anzahl der Datensätze erlaubt. Gerade wenn Sie gezielt einzelne Datensätze ändern wollen und dann hier eine unerwartet große Zahl sehen, wird deutlich, dass Sie die Kriterien für die WHERE-Bedingung vergessen haben.

Wenn Sie diese Abfrage bestätigen, sind die Änderungen unwiderruflich durchgeführt.

Abbildung 6.11 Kontrollabfrage Aktualisierung: Sind wirklich so viele Datensätze betroffen?


Microsoft Office Access
Sie beabsichtigen, 11 Zeile(n) zu aktualisieren.
Sobald Sie auf 'Ja' geklickt haben, können Sie die Änderungen nicht mehr mit dem Befehl 'Rückgängig' zurücknehmen.
Sind Sie sicher, dass Sie diese Datensätze aktualisieren möchten?
Ja
Nein

6.2.3 Zusammenfassung

Mit der SQL-Anweisung UPDATE können einzelne oder Gruppen von Datensätzen bis hin zu ganzen Tabelleninhalten geändert werden. Die Syntax der UPDATE-Anweisung lautet im Allgemeinen:

UPDATE-Syntax

UPDATE *tabellenname*

SET *feldname1=ausdruck [{, feldname = ausdruck}]*

[WHERE *bedingungsliste];*

Bevor Sie eine UPDATE-Anweisung an den SQL-Interpreter senden, sollten Sie sie als einfache Auswahlabfrage mit SELECT testen. Sie können die funktions-

fähige Abfrage dann in eine UPDATE-Anweisung umbauen. Wichtig ist in jeder UPDATE-Anweisung, die Verwendung der WHERE-Klausel zu überprüfen. Ohne WHERE-Klausel werden alle Datensätze der Tabelle geändert.

6.2.4 Update-Übungen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine UPDATE-Anweisung.

Hinweis: Nutzen Sie in der Datenbank gegebenenfalls eine Kopie der Tabellen, damit Sie die ursprünglichen Tabellen nicht zerstören. In MS Access können Sie die Tabellen dafür im Datenbankfenster kopieren, in den anderen Systemen können Sie die ursprünglichen Tabellen neu laden (siehe Kapitel 3).

1. Die Familiennamen „Weiss“ sind irrtümlich alle ohne ß eingefügt worden und sollen in „Weiß“ umbenannt werden. Ändern Sie alle betroffenen Datensätze. Hinweis: Sie können die Änderung mit einem SELECT überprüfen. (Ü6.2.1)
2. Machen Sie die eben gemachten Änderungen mit einer neuen UPDATE-Anweisung wieder rückgängig. (Ü6.2.2)
3. Erhöhen Sie den Stundensatz aller Dozenten, die 15,- € oder weniger verdienen um 10 %. (Ü6.2.3)
4. Machen Sie die eben gemachten Änderungen mit einer neuen UPDATE-Anweisung wieder rückgängig. (Ü6.2.4)

6.3 Datensätze löschen (DELETE)

6.3.1 DELETE-Grundlagen

Die dritte Möglichkeit, den Inhalt von Tabellen zu ändern, besteht im Löschen von Datensätzen. Gelöscht werden wie beim Einfügen grundsätzlich komplette Datensätze. Sollen einzelne Felder „gelöscht“ werden, kann dies über das Ändern auf einen NULL-Wert geschehen.

Beispiel

In Abschnitt 6.1 wurde mit der folgenden SQL-Anweisung ein neuer Datensatz in die Personentabelle eingefügt.

```
INSERT INTO tbPerson(
   PID, Familienname, Vorname, PLZ, Ort, Strasse, Geburtsdatum
)
VALUES(
   '51','Gerhardt',NULL,'29221',DEFAULT,'Zöllnerstraße 12',
   '1955-02-24'
);
```

Diese Zeile soll jetzt wieder gelöscht werden. Bevor Sie dies tun, beachten Sie den dringenden Rat, vor jeder DELETE-Anweisung die Anweisung als normale SELECT-Anweisung zu testen, da ein „Rückgängig“ nicht möglich ist.

Um gezielt einen einzelnen Datensatz auszuwählen, der gelöscht werden soll, ist eine WHERE-Klausel auf den Primärschlüssel sinnvoll. Testen Sie mit dieser Anweisung, ob es sich um den gewünschten Datensatz handelt:

```
SELECT *
FROM tbPerson
WHERE PID=51;
```

Im Ergebnis erhalten Sie hoffentlich den richtigen Datensatz. Wir stellen die SELECT-Anweisung dann auf die gewünschte DELETE-Anweisung um:

```
DELETE
FROM tbPerson
WHERE PID=51;
```

Sie sehen, dass sich der Aufwand wie auch die Fehleranfälligkeit dabei in Grenzen hält.

DELETE-Syntax

Die allgemeine Syntax für die DELETE-Anweisung lautet:

DELETE

FROM *tabellenname*

[WHERE *bedingungsliste];*

Die Bedingungsliste mit der WHERE-Klausel entspricht der Syntax der SELECT-Anweisung. Damit können Sie gezielt einen Datensatz für eine DELETE-Anweisung auswählen, aber natürlich auch Bedingungen angeben, die dazu führen, dass viele Datensätze gelöscht werden. Alle nach Auswertung der WHERE-Klausel übrig bleibenden Datensätze werden aus der Tabelle gelöscht.

Weiteres Beispiel

Angenommen, Sie wollen alle Kursteilnehmer des Kurses „CE24“ aus der Tabelle der Kursteilnehmer tbKursbesuche löschen. Dann lautet die Syntax:

```
DELETE
FROM tbKursbesuche
WHERE KID = 'CE24';
```

Es werden alle Datensätze gelöscht, die die WHERE-Klausel erfüllen.

6.3.2 Alle Datensätze löschen (TRUNCATE)

Löschen aller Datensätze

Über die WHERE-Klausel steuern Sie, welche Datensätze gelöscht werden. Es werden immer alle Datensätze gelöscht, die die angegebene WHERE-Klausel erfüllen. Das bedeutet in der Konsequenz auch, dass bei fehlender WHERE-Klausel – wenn diese beispielsweise einfach vergessen wird – alle Datensätze aus einer Tabelle gelöscht werden:

```
DELETE
FROM tbKursbesuche;
```

Diese einfache Anweisung hat also unter Umständen die fatale Folge, dass Ihre Datensätze alle verschwunden sind. Wenn Sie es ausprobieren: Sie können die Tabelle wie in Kapitel 3 beschrieben anschließend durch einen Import wiederherstellen – beziehungsweise auf eine Kopie der MS Access-Datenbank Kurse zurückgreifen.

Es wird allerdings nicht die Tabelle selbst gelöscht, dazu gibt es die DROP-Anweisung, die nicht nur die Datensätze einer Tabelle, sondern auch deren Struktur löscht (siehe Kapitel 8).

Im Normalfall wollen Sie aber mit einer DELETE-Anweisung nicht alle Datensätze und erst recht nicht die Tabellenstruktur löschen.

TRUNCATE

Wenn Sie allerdings wirklich alle Datensätze einer Tabelle löschen wollen, so können Sie dies prinzipiell mit einem DELETE ohne WHERE-Klausel erreichen. Von einigen Datenbanken wird als Alternative zum Löschen aller Datensätze einer Tabelle die TRUNCATE-Anweisung angeboten. Auch TRUNCATE löscht nicht die Tabellenstruktur selbst, sondern „nur“ alle Datensätze in der Tabelle. Die TRUNCATE-Anweisung ist allerdings beim Löschen zumeist deutlich performanter als eine DELETE-Anweisung für die gesamte Tabelle.

Tipp

Wenn Sie alle Datensätze in einer Tabelle löschen wollen, kann die DELETE-Anweisung sehr zeitaufwendig sein, da die Datensätze alle einzeln gelöscht werden. Eine Alternative stellt hier die TRUNCATE-Anweisung dar, die alle Datensätze ohne aufwendige Protokollierung „am Stück“ löscht.

TRUNCATE FROM *tabellenname;*

Viele Datenbanken – so auch MySQL und Oracle – unterstützen diese Anweisung, MS Access hält sich hier leider zurück.

6.3.3 Besonderheiten des DELETE bei MS Acess

MS Access

MS Access bietet in der grafischen Oberfläche eine Möglichkeit, auch Löschabfragen zu stellen. Das Vorgehen entspricht dabei weitgehend dem beim INSERT und UPDATE beschriebenen Verfahren. Sie sollten hier allerdings noch einmal besondere Vorsicht walten lassen. Am besten legen Sie zunächst eine „normale“ SELECT-Abfrage – in MS Access Auswahlabfrage genannt – an. Mit dieser Abfrage können Sie dann testen, ob Sie mit Ihren Kriterien wirklich die gewünschten Datensätze ermitteln können, die mit einer Löschabfrage entfernt werden sollen.

Wenn Sie beispielsweise alle Teilnehmer des Kurses „CE24“ löschen wollen, wählen Sie nur das Feld KID aus der Tabelle tbKursbesuche aus. Als Kriterium tragen Sie ="CE24" ein. Diese Abfrage können Sie jetzt wie gewohnt ausführen und erhalten die gewünschten fünf Datensätze. Bei Bedarf können Sie weitere Attribute hinzunehmen. Dies ist besonders während der Testphase sinnvoll, wenn Sie prüfen wollen, ob Sie wirklich die gewünschten Datensätze erhalten. Für die eigentliche Löschabfrage benötigen Sie nur die Felder, die Teil der WHERE-Klausel werden sollen.

Sie können jetzt einfach über den Menüpunkt ABFRAGE/LÖSCHABFRAGE auf eine Löschabfrage umschalten. MS Access generiert dann daraus eine DELETE-Anweisung:

```
DELETE tbKursbesuche.KID
FROM tbKursbesuche
WHERE (tbKursbesuche.KID="CE24");
```

Sie sehen, dass MS Access die ausgewählten Felder zusätzlich in die DELETE-Anweisung aufnimmt. Das ist zwar nicht notwendig und eigentlich auch nicht vorgesehen, funktioniert aber. Wenn Sie die Abfrage dann ausführen,

werden die fünf Datensätze, die wir bereits mit der Auswahlabfrage ermittelt hatten, nach einer Sicherheitsabfrage tatsächlich gelöscht. Die MS Access-Löschabfrage ist also die grafische Umsetzung der DELETE-Anweisung.

6.3.4 Übungen zur DELETE-Anweisung

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine DELETE-Anweisung.

Hinweis: Nutzen Sie in der Datenbank gegebenenfalls eine Kopie der Tabellen, damit Sie die ursprünglichen Tabellen nicht zerstören. In MS Access können Sie die Tabellen dafür im Datenbankfenster kopieren, in den anderen Systemen können Sie die ursprünglichen Tabellen neu laden (siehe Kapitel 3).

1. Löschen Sie alle Datensätze, die in den Übungen des Abschnitts 6.1.5 eingefügt wurden. (Ü6.3.1)
2. Es sollen alle Kursteilnehmer aus der Tabelle `tbKursbesuche` entfernt werden, die einen oder mehr Fehltage haben. (Ü6.3.2)
3. Es sollen alle Kursteilnehmer aus der Tabelle `tbKursbesuche` entfernt werden, die mit einem Gutschein bezahlen. (Ü6.3.3)
4. Löschen Sie alle Kursthemen aus der Tabelle `tbKursthema`, bei denen aus der Kursbeschreibung zu entnehmen ist, dass der Kurs etwas mit Programmen oder Programmierung zu tun hat. (Ü6.3.4)
5. Löschen Sie alle Dozenten aus der Tabelle `tbDozent`, die entweder selbstständig sind oder als Titel „Meister“ tragen. (Ü6.3.5)

7

Datenbanken modellieren

Datenbanken sind ein sinnvolles Hilfsmittel zur Verwaltung umfangreicher strukturierter Datenbestände. Mit SQL-Abfragen, wie wir sie in den vorherigen Abschnitten behandelt haben, können Informationen schnell gewonnen und genutzt werden. Voraussetzung ist allerdings, dass die Daten in der Datenbank in einer „vernünftigen" Struktur vorliegen.

7.1 Das 3-Ebenen-Modell

7.1.1 Anforderungen an das Datenbankmodell

Im Folgenden werden die wesentlichen Aspekte der Datenbankmodellierung betrachtet, an einigen Stellen auch vereinfacht. So musste beispielsweise bei der Modellierung mehr auf das Ergebnis als auf den Prozess eingegangen werden. Bestimmte Modellierungsaspekte wie hierarchische Beziehungen und einige andere Konstrukte werden nicht weiter diskutiert. Bedenken Sie aber bitte, dass dies eine Einführung in SQL, nicht in den Datenbankentwurf ist. Dieses Kapitel ist hier enthalten, um mehr Hintergrundverständnis für die Entstehung des Datenbankschemas zu schaffen, das im nächsten Kapitel dann mit SQL realisiert wird.

Vernünftiges Modell

Die Struktur einer relationalen Datenbank wird von ihrem Modell, von den Tabellen, den Feldern, den Primär- und Fremdschlüsseln bestimmt. Was ist aber eine vernünftige Struktur? Welches Modell ist richtig, welches falsch? Immer wenn man mir diese Frage gestellt hat, habe ich geantwortet, dass es keine richtigen und falschen Modelle gibt, nur nutzbare und weniger nutzbare Modelle.

Ein vernünftiges Modell bedeutet daher zunächst, dass es sich zu dem Zweck eignet, für den eine Datenbank gedacht ist. Der Ausgangspunkt des Datenbankmodells sind die betrieblichen Prozesse, die anschließend unterstützt werden sollen.

Weil Datenbanken, wenn sie erst einmal in Betrieb sind und Abfragen und Änderungsmechanismen realisiert sind, nur mit erheblichem Aufwand geändert werden können, bedarf es eines sorgfältigen Datenbankentwurfs, der dann mit SQL umgesetzt wird.

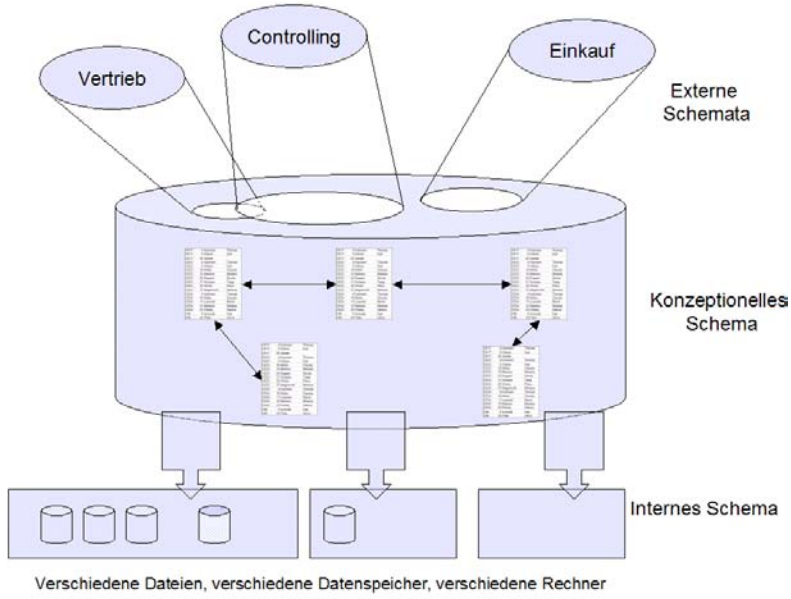
7.1.2 Die drei Ebenen des Datenbankmodells

ANSI/SPARC

Vor dem eigentlichen Entwurf soll kurz auf das 3-Ebenen-Modell eingegangen werden. Dieses Modell wird auch als ANSI/SPARC-Modell bezeichnet und wurde 1978 in seiner detaillierten Fassung vom Standard Planning And Requirement Commitee (SPARC) des American National Standards Institute (ANSI) vorgelegt. Die Architektur spielt bis heute die zentrale Rolle für die Hersteller und Betreiber von Datenbanksoftware und Zugriffssystemen. Sie bildet in gewisser Weise auch die Basis für die Standardisierung von SQL, da hier mit dem konzeptionellen (logischen) Schema eine Ebene geschaffen wurde, die von der internen Speicherung abstrahiert und die verschiedenen Anwendersichten integriert. Damit war die Basis für die Schaffung einer standardisierten Sprache geschaffen.

Das 3-Schichten-Modell unterscheidet die physische Struktur der Datenspeicherung, das logische Modell und die externen Sichten auf die Datenbank. Jede dieser drei Ebenen bedarf eines Entwurfes, wenn man sich in der Darstellung auch zumeist auf das logische Modell des konzeptionellen Schemas beschränkt.

Abbildung 7.1
Das 3-Schichten-Modell einer Datenbank



Internes Schema

Die physische Struktur des internen Schemas ist eine Sache der konkreten Datenbank und deren Umgang mit den vom Betriebssystem bereitgestellten Schnittstellen zur Verwaltung des physikalischen Speicherplatzes, beispielsweise auf einer Festplatte. Hier geht es darum, wie Speicherplatz reserviert wird, ob beispielsweise aus Sicht des Betriebssystems eine große Datei angelegt wird, deren Inhalt und Struktur nur das Datenbankmanagementsystem kennt und bearbeitet, oder ob mehrere kleinere Dateien verwaltet werden. Physisch können die Daten auch über mehrere Speicher verteilt werden, bei größeren Datenbanken ist das oft sogar performanter. Die Daten können auch auf verschiedene Rechner verteilt werden. Dies ist Sache des Datenhaltungssystems.

Optimierung

Idealerweise sollte der Benutzer bei der Bearbeitung der Datenbank möglichst wenig mit diesen internen Strukturen zu tun haben (physische Datenunabhängigkeit). In der Realität haben aber beispielsweise Indizes, Tablespaces und andere Zugriffsoptimierungen Einfluss auf die physikalische Organisation der Daten und damit insbesondere auf die Performance der Datenbank. So trifft in den meisten Datenbanken der sogenannte Query-Optimizer die Entscheidung über die Art der Ausführung eines SQL-`SELECT`. Durch geeignete Bereitstellung von Indizes, Zugriffspfaden und anderen Zugriffshilfen kann dies durch den Datenbankadministrator im Datenbankentwurf beeinflusst werden. Diese Bereitstellung erfolgt ebenfalls durch SQL-Befehle wie `CREATE INDEX` oder `CREATE TABLESPACE`, auf die später einzugehen ist. Leider sind viele Mechanismen bei den verschiedenen Datenbankmanagementsystemen recht unterschiedlich realisiert, sodass eine allgemeine Beschreibung hier den Rahmen sprengen würde. Die entsprechenden Mechanismen sind den Handbüchern der Datenbanksysteme zu entnehmen.

Konzeptionelle Ebene

Das logische Schema der konzeptionellen Ebene beinhaltet die Sicht auf die Tabellen, Felder und Schlüssel, also im Wesentlichen die bisher im Zusammenhang mit der SQL-Syntax betrachteten Elemente. Es stellt die wesentliche Basis für den Zugriff mit SQL auf die Datenbank dar. Jeder `SELECT`-Befehl, jedes `INSERT`, `UPDATE`, `DELETE` und alle weiteren hier bisher vorgestellten Begriffe greifen im Wesentlichen auf diese Strukturen zurück.

Externe Datenschemata

Jeder professionelle Datenbankanwender wird allerdings nicht unmittelbar auf Tabellen zugreifen, wie wir dies bisher getan haben. Vielmehr wird er einen indirekten Zugriff auf die Datenbank bevorzugen, bei dem sogenannte *VIEWs* verwendet werden (siehe hierzu Kapitel 8), die das externe Schema bilden. Ein `VIEW` stellte dabei eine Art virtuelle Tabelle dar. Aus Sicht einer `SELECT`-Anweisung verhält sich ein `VIEW` grundsätzlich wie eine Tabelle, es kann also normal mit einem `SELECT` auf ein `VIEW` zugegriffen werden:

```
SELECT * FROM viewname WHERE ...;
```

Seine Daten erhält der `VIEW` allerdings nicht direkt aus einer Tabelle, sondern er greift seinerseits durch einen eigenen `SELECT`-Befehl auf Tabellen (oder andere `VIEW`s) zu, um die so gewonnenen Daten dann selbst wieder wie eine Tabelle präsentieren zu können.

Die genaue Beschreibung der entsprechenden SQL-Kommandos finden Sie in Abschnitt 8.6.

7.1.3 Der Weg zum Datenbankmodell

Den zentralen Punkt für die Erstellung der Datenbank und letztlich den Einsatz von SQL stellt das logische Modell der konzeptionellen Ebene dar, auf das unmittelbar in Form von Tabellen oder mittelbar über einen `VIEW` zugegriffen wird. Das logische Modell selbst ist das Endergebnis eines Erstellungsprozesses.

Wird festgestellt, dass für die Verwaltung der Daten eines bestimmten Problembereiches eine Datenbank benötigt wird, beginnt der Planungsprozess der Datenbank, der sogenannte Datenbankentwurf. Dabei handelt es sich um einen Prozess, bei dem festgestellt wird, welche Daten von welchen Anwendern benötigt werden.

Der Datenentwurfsprozess ist wiederum Bestandteil eines größeren Ganzen, bei dem neben den Daten auch die benötigte Funktionalität berücksichtigt wird. Dieser heute weitgehend objektorientierte Software-Entwicklungsprozess beginnt zumeist mit einer Anforderungsanalyse. Die eigentliche Anforderungsanalyse ist dabei bereits im Vorfeld der Datenanalyse anzusiedeln. Sie kann mithilfe der UML (Unified Modeling Language), mit Ereignis-Prozess-Ketten (EPK) oder klassisch in Form von schriftlichen Pflichten-/Lastenheften erfolgen. Danach werden je nach Methode in einem phasenorientierten Vorgehen, in einem Spiralmodell, mit Prototyping oder Extreme Programming die weiteren Entwicklungsschritte durchgeführt.

Aus Datenbanksicht sind im Rahmen des Entwicklungsprozesses dabei verschiedene Aspekte zu beachten:

- Welche Datensichten werden von verschiedenen Anwendern und Programmen auf die Datenbank benötigt (externe Schemata, `VIEWS`)?
- Wie können sie zu einer gemeinsamen konzeptionellen **Struktur (konzeptionelles Schema)** zusammengefasst werden?
- Wie soll die logische Struktur aussehen, in welcher Form werden also die Daten gespeichert, und wie soll mit SQL darauf zugegriffen werden?
- Wie muss die **physische Struktur (internes Schema)** der Datenbank angepasst werden, um die Zugriffe zu optimieren?

Datenbanken werden – wie Software auch – in mehreren aufeinanderfolgenden Phasen entwickelt. Das Ziel ist es, von einer abstrakten und oft nur sprachlich unklar formulierten Anforderung der Nutzer zu einer exakten für den Rechner verständlichen Formulierung der Struktur zu kommen.

Bei der reinen Entwicklung und Nutzung einer Datenbank (ohne Programmierung) können Sie fünf Phasen voneinander unterscheiden:

Tabelle 7.1
Idealtypische Vorgehensweise bei der Erstellung einer Datenbank

Phasenname	Tätigkeiten	Ergebnisse
Anforderungsanalyse	In der Anforderungsanalyse werden die Anforderungen aller Benutzer an die neue Datenbank zusammengetragen. Es kann auf UML oder andere Vorarbeiten zurückgegriffen werden. Diese Anforderungen werden meist nach bestimmten Kriterien klassifiziert, beispielsweise nach Abteilungen oder Benutzergruppen. Wichtig ist, dass festgelegt wird, welche Daten gespeichert werden sollen (was zu speichern ist) und wie die Daten zu bearbeiten sind.	Pflichtenheft
Konzeptioneller Entwurf	Am Ende dieser Phase liegen die Sichten und das konzeptionelle Gesamtschema (meist als Entity-Relationship-Modell) vor. Beim Entwurf können Sie verschiedene Vorgehensweisen verwenden. Die externen Sichten müssen nicht zwingend entworfen werden. Werden sie entworfen, haben Sie mehrere Möglichkeiten. Entweder Sie entwerfen zuerst die externen Sichten und fügen diese dann zu einem konzeptionellen Schema zusammen oder umgekehrt.	Konzeptionelles Schema als ERM oder Klassendiagramm
Logischer Entwurf	Das konzeptionelle Schema wird in das logische (relationale) Schema der Datenbank überführt. Zusätzlich wird die Datenbank qualitativ für das relationale Schema optimiert, zumeist durch Normalisierung. Bevor der logische Entwurf durchgeführt werden kann, muss festgelegt werden, für welches DBMS die Datenbank aufgebaut werden soll.	Normalisiertes logisches Schema mit Tabellen, Feldern und Beziehungen
Verfeinerung des logischen Entwurfs	Nun kann der logische Entwurf in Hinblick auf häufige oder bevorzugte Abfragen, die in den Anforderungen formuliert wurden, optimiert werden. Dabei werden Erweiterungen und gegebenenfalls Änderungen am relationalen Schema durchgeführt, beispielsweise durch das Einfügen von Indizes.	Logisches Schema mit physischen Eigenschaften wie Indizes, Datentypen und andere

Tabelle 7.1 (Forts.) Idealtypische Vorgehensweise bei der Erstellung einer Datenbank

Phasenname	Tätigkeiten	Ergebnisse
Implementierung	In der letzten Entwurfsphase erfolgt die Definition des internen Schemas. Es werden geeignete Speicherstrukturen und Zugriffsmechanismen darauf festgelegt. Ein wichtiger Aspekt ist auch das Laufzeitverhalten des DBMS, das durch einen effizienten Zugriff auf die relevanten Daten verbessert werden kann. In der Datendefinitionssprache (DDL) des DBMS werden nun das physische Schema (Tablespaces, Indizes), das logische Schema (Tabellen) und das externe Schema (`VIEW`s) implementiert. Die Festlegung der Zugriffsrechte erfolgt ebenfalls in dieser Phase.	Schema wird in der Datenbank implementiert.

7.2 Das Entity-Relationship-Modell (ERM)

Das Entity-Relationship-Modell, kurz als ERM bezeichnet, steht zumeist am Anfang jedes Datenbankentwurfs. Beim konzeptionellen Entwurf geht es darum, aus den meist unstrukturierten Informationen, die man über den Inhalt einer Datenbank hat, ein Modell zu gewinnen, das

- konkret und strukturiert genug ist, um eine Datenbank daraus erstellen zu können,
- allgemein genug ist, um von Nicht-IT-Fachleuten verstanden zu werden.

Die bekannteste Methode zur Beschreibung eines konzeptionellen Schemas ist das Entity-Relationship-Modell von Peter Chen (1976), das historisch also jünger als das relationale Modell ist.

Am Anfang ist das Modell

Das Modell steht also am Anfang der Datenbankentwicklung. Das Ziel ist es, ein richtiges Abbild der Realität der künftigen Anwender zu finden. „Richtig“ bedeutet dabei, dass die Daten einfach abgelegt, geändert und insbesondere abgefragt werden können. Gerade der letzte Aspekt ist wichtig. Zwar bieten relationale Datenbanken eine große Flexibilität und Funktionalität, aber das kann in der Praxis auch problematisch werden. So wird eine Datenbank, die bereits am Anfang nur schwer die Abfragen der Anwender beantworten kann und eine schlechte Performance aufweist, nur schwer Akzeptanz bei den Anwendern finden.

Definition Modell

Unter einem Modell ist eine **vereinfachte** Abbildung der Realität oder einer Idee zu verstehen, die alle für den jeweiligen Zweck wesentlichen Aspekte enthält.

Wir kennen in der Realität viele Modelle. So gibt es allein für unsere Erde zahlreiche Modelle, um die Komplexität des Ganzen zu reduzieren. Wenn Sie

mit dem Fahrzeug unterwegs sind, ist eine Autokarte von Deutschland hilfreich. Die Karte beinhaltet die für den Zweck Autofahrt wesentlichen Angaben. Insbesondere werden Sie alle Straßen, Hinweise auf Orientierungshilfen wie Bahnlinien, Flüsse und Ähnliches finden. Dagegen beinhaltet die Karte keine Hinweise auf Bodenschätze, jährliche Niederschlagsmengen, Durchschnittstemperaturen oder Bevölkerungsdichten. Hiervon wird abstrahiert, da diese Informationen für den Zweck nicht notwendig sind. Wollen Sie eine Fernreise mit dem Auto unternehmen, ist dagegen eine Karte mit dem Autobahn- und Fernstraßennetz Europas hilfreich, für eine Radtour in der Lüneburger Heide die entsprechende Fahrradkarte. Politische Karten, geografische Erdkarten und andere Karten können für andere Zwecke wiederum nützlich sein.

Wir kennen viele andere Modelle von Modelleisenbahnen über Auto- und Flugzeugmodelle bis zu makroökonomischen Modellen wie beispielsweise von John Maynard Keynes, die bestimmte Aspekte des wirtschaftlichen Handelns beschreiben.

Entity-Relationship-Modell

Das Entity-Relationship-Modell (ERM) ist das bekannteste und meistverwendete grafische Hilfsmittel, um ein Modell des Datenhaushaltes für einen bestimmten Zweck zu schaffen. Es wird darüber hinaus auch in anderen Bereichen der Informatik eingesetzt, in denen Ausschnitte der realen Welt modelliert werden. Es ist unabhängig von einem bestimmten Datenbanksystem und unterliegt nicht den Einschränkungen, die sich durch die Implementierung in einer Datenbank wie MS Access, MySQL oder Oracle ergeben können.

Entität, Beziehung und Attribut

Das ERM ermöglicht es, die konzeptionellen Entwürfe einer Datenbank auf leicht verständliche Art grafisch darzustellen. Die zwei Grundbausteine des ERM sind die Entitäten (engl. entity: Sache, Ding) und die Beziehungen(engl.: relationship). Entitäten (und seltener Beziehungen) haben Attribute (engl.: attribute), die ihre Eigenschaften beschreiben.

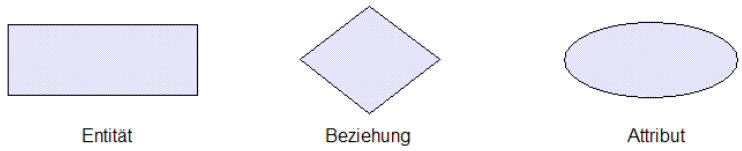
Das ERM ist also ein Modell, das eine strukturierte Darstellung der Informationen für einen bestimmten Zweck in grafischer Form mithilfe von Entitäten (Entity), Beziehungen (Relationship) und Attributen erlaubt. Von der physischen Umsetzung der Speicherung und Verwaltung der Informationen wird abstrahiert.

Zur leichteren Veranschaulichung können Sie sich vorstellen, dass aus einer Entität in der Datenbank eine Tabelle wird. Die Beziehungen sorgen dafür, dass die Tabellen miteinander verbunden werden können, also die Primärschlüssel-Fremdschlüssel-Verbindungen definiert und für die verschiedenen Arten des `JOIN` verwendet werden können. Das ERM ist aber noch kein logisches Schema für eine Datenbank, kann also noch nicht direkt in MySQL, Oracle, MS Access oder ein anderes Datenbankmanagement eingegeben werden. Es beschreibt auch nicht, was mit den Informationen gemacht wird, beschreibt also keine Funktionen, Programme oder Geschäftsabläufe. Es bietet aber die Basis für eine spätere Umsetzung in eine Datenbank.

Es gibt verschiedene Varianten des ERM, die sich durch die Art der Darstellung und die Menge der enthaltenen Informationen unterscheiden. Wir wollen hier die ursprüngliche Schreibweise von Chen verwenden.

Die drei Grundelemente Entität (Entity), Beziehung (Relationship) und Attribut (Attribut) sind in Abbildung 7.2 dargestellt. Die grafische Darstellung erfolgt als Rechteck (Entität), Raute (Beziehung) und abgerundetes Rechteck oder Ellipse bzw. Kreis (Attribute).

Abbildung 7.2
Grafische Elemente eines Entity-Relationship-Modells



Entwicklung eines Modells

7.2.1 Entitäten

Die Entwicklung eines Entity-Relationship-Modells beginnt meistens mit der Suche nach den Entitäten. Will man aus einer Beschreibung und aus Gesprächen mit dem Anwender ermitteln, welche Entitäten für die Lösung seines Problems infrage kommen, ergeben sich aus diesen Gesprächen und Niederschriften oft eine Reihe von Anhaltspunkten. Gute Kandidaten für Entitäten sind:

- Substantive, die oft verwendet werden und zum Problemumfang gehören,
- Substantive, die Dinge oder Ideen beschreiben, die Sie durch weitere Eigenschaften genauer beschreiben können,
- Substantive, von denen mehrere reale Objekte existieren,
- Substantive, die Aktivitäten beschreiben oder die später leicht aus anderen Informationen berechenbar sind,
- Substantive, deren Instanzen Sie mithilfe einer oder mehrerer Eigenschaften eindeutig identifizieren können (Schlüssel).

7.2.2 Attribute (Eigenschaften)

Attribute

Attribute – oft auch als „Eigenschaften“ bezeichnet – charakterisieren eine Entität. Ein Attribut besitzt einen Namen und einen Datentyp. In der grafischen Darstellung werden die Attribute als abgerundete Rechtecke, als Ellipsen oder Kreise dargestellt. Diese sind über ungerichtete Kanten mit der Entität verbunden. Bei den Attributen wird zwischen beschreibenden Attributen (den anwendungsspezifischen Eigenschaften) und identifizierenden Attributen (den Schlüsseln zur eindeutigen Identifikation) unterschieden.

Schlüssel

Ein Schlüssel setzt sich aus einem oder mehreren Attributen zusammen. Es sollten so wenige wie möglich (Minimalitätsforderung), aber so viele wie nötig sein. Das ist wichtig, da so beim späteren Erstellen der Datenbank entschieden werden kann, ob diese Schlüssel direkt verwendet werden können

oder ob künstliche Schlüssel vergeben werden sollten und die hier definierten Schlüssel über zusätzliche Bedingungen (Constraints) umgesetzt werden sollen. Künstliche Schlüssel sollten im ER-Modell nach Möglichkeit noch nicht vergeben werden. Ist unter den vorhandenen Attributen und Attributkombinationen keine, die als Schlüssel eingesetzt werden kann, sollte das Modell noch einmal überprüft werden. Nur in sehr seltenen Fällen muss bereits hier ein künstlicher Schlüssel vergeben werden. Sollten die Anwender einen neuen Schlüssel wünschen, weil dies bei der Gestaltung der Abläufe zu mehr Klarheit führt, kann dieser natürlich ergänzt werden. Der Schlüssel entspricht dann einer Anwenderanforderung.

Schlüsselkandidat

Es können für eine Entität mehrere Schlüssel existieren, die dann als Schlüsselkandidaten bezeichnet werden. So kann ein Haus beispielsweise alternativ durch

- Ort, Straße und Hausnummer,
- die Gemeinde und die Grundbucheintragung oder
- die geografische Länge und Breite

identifiziert werden. In solchen Fällen muss die Entscheidung getroffen werden, welcher Schlüsselkandidat als Primärschlüssel verwendet werden soll.

Primärschlüssel

Der Primärschlüssel ermöglicht die eindeutige Identifizierung einer Entität dadurch, dass sein Wert in einer Entitätsmenge (entspricht der Menge der Datensätze) nur ein einziges Mal vorkommt. Eine Entität kann mehrere Schlüsselkandidaten besitzen, die für bestimmte Abfragen oder Sortierungen benötigt werden, aber nur einen Primärschlüssel. Die Situation ist also einfach. Alle möglichen Kombinationen eindeutiger Attribute sind Schlüsselkandidaten. Ein Schlüsselkandidat wird als Primärschlüssel ausgewählt. Alle anderen Schlüsselkandidaten sind Alternativschlüssel. In der grafischen Darstellung werden Primärschlüssel durch unterstrichene Attribute markiert.

7.2.3 Domänen

Eine Domäne beschreibt den zulässigen Wertebereich eines Attributes. Im Normalfall handelt es sich dabei um einen Datentyp. Ein Datentyp hat einen festen Wertebereich, beispielsweise hat der Datentyp `INTEGER` einen festen Zahlenbereich, wie in Kapitel 5 beschrieben. Es können in einem Attribut dieses Typs nur Zahlen aus diesem Wertebereich gespeichert werden, also ganze Zahlen in einer bestimmten Größenordnung.

In einigen Fällen ergeben sich aus dem Kontext eines Attributes weitere Einschränkungen des Wertebereiches. Das können fest vorgegebene Wertemengen sein (z.B. Januar, Februar ...), Bereiche (z.B. von 0 bis 999, von A bis G) oder Mengen solcher Werte.

Beispiel

In der Kursdatenbank werden als Zahlungsart nur „Gutschein“, „bar“ oder „Überweisung“ zugelassen. Es wird zwar ein alphanumerischer Datentyp (`CHARACTER`) verwendet, tatsächlich ist der Wertebereich aber viel kleiner als die Menge aller Buchstabenkombinationen, die Sie aufgrund des Datentyps bilden könnten.

Derartige Domänen sollten der Beschreibung der Attribute bereits frühzeitig, hier im Rahmen des ER-Modells, beigefügt werden, da sie den Anwendern oft bekannt sind und so bereits bei der Erzeugung des Datenbankschemas berücksichtigt werden können.

Hier entsteht ein Zwiespalt. Einerseits möchte man sich nicht zu früh auf einen bestimmten physischen Datentyp festlegen, da man eine frühzeitige Festlegung auf ein bestimmtes Datenbanksystem vermeiden möchte. Andererseits kann es mühsam werden, künstlich Domänen zu beschreiben, von denen man schon genau weiß, was daraus werden soll. In größeren Zusammenhängen ist die Mühe, hier eine genaue fachliche Festlegung zu treffen, trotzdem sinnvoll. Hier wollen wir aber aus Vereinfachungsgründen schnell auf die bekannten Datentypen zurückgreifen.

7.2.4 Beziehungen

Durch Beziehungen werden die Abhängigkeiten und Zusammenhänge zwischen Entitäten ausgedrückt. Eine Beziehung wird grafisch durch eine Raute dargestellt, die durch (im Normalfall) zwei Kanten mit den Entitäten verbunden ist, die assoziiert werden sollen. In der Raute kann der Name der Beziehung stehen. Beziehungen können zusätzlich durch Attribute beschrieben werden, beispielsweise kann die Beziehung „arbeitet an“ zwischen den Entitäten „Mitarbeiter“ und „Projekt“ um zwei Attribute ergänzt werden, die angeben, in welcher Tätigkeit ein Mitarbeiter an einem Projekt mitarbeitet und zu wie viel Prozent.

Grad einer Beziehung

In der Regel gehören zu einer Beziehung zwei Entitäten. Es können aber auch mehrere Entitäten assoziiert werden. Die Anzahl der an einer Beziehung beteiligten Entitäten wird als Grad der Beziehung bezeichnet. Wird beispielsweise ein Produkt aus mehreren Bauteilen verschiedener Lieferanten zusammengesetzt, hat die Beziehung die Form (`PRODUKT`, `BAUTEIL`, `LIEFERANT`). Der Grad der Beziehung ist dann 3 (ternäre Beziehung).

Definition

Wechselwirkungen und Abhängigkeiten zwischen Entitäten werden durch Beziehungen (Assoziationen) dargestellt. Eine Beziehung gilt im Normalfall zwischen genau zwei Entitäten. So lässt sich eine Ehe als Beziehung „ist verheiratet“ zwischen den beiden Entitäten `EHEMANN` und `EHEFRAU` definieren:

```
EHEMANN ist verheiratet mit EHEFRAU
```

Dieselbe Beziehung lässt sich aber auch verallgemeinern, indem Sie Ehefrau und Ehemann zu einer Entität `PERSON` zusammenfassen. Es ergibt sich dann die Beziehung

```
PERSON ist verheiratet mit PERSON
```

Hinter einer solchen Beziehung verbergen sich dann konkrete Zuordnungen, beispielsweise:

- Herr Meyer ist verheiratet mit Frau Meyer.
- Herr Schulz ist verheiratet mit Frau Schulz.
- Herr Müller ist verheiratet mit Frau Müller-Lüdenscheid.

1:1-, 1:n- und n:n-Beziehungen

Eine Beziehung bezieht sich also normalerweise auf zwei oder eine Entität in zwei Rollen. Wesentlich für eine Beziehung ist dabei, dass es sich immer um einen Zusammenhang zwischen Entitäten handeln muss, sie also nicht als Attribut einer Entität aufgefasst werden kann.

Tabelle 7.2 Grundtypen von Beziehungen

Beziehung	Beschreibung	Beispiele
1:n	„eins-zu-n“ oder „eins-zu-viele“ Normalfall. Eine Ausprägung der einen Entität kann maximal einer Ausprägung der anderen Entität zugeordnet werden, umgekehrt aber können es viele sein.	**angestellt** Der `MITARBEITER` „Meier“ ist angestellt bei einer `FIRMA` „DatenWelt“. Die `FIRMA` „DatenWelt“ hat einen oder viele `MITARBEITER`, beispielsweise „Meier“, „Hoffmann“ ...
1:1	„eins-zu-eins“ Eine Ausprägung der einen Entität kann nur maximal einer Ausprägung der anderen Entität zugeordnet werden, und umgekehrt.	**verheiratet (christliche Variante)** Eine `PERSON` ist mit keiner oder einer `PERSON` verheiratet. Das Ganze gilt in beiden Richtungen.
n:n	„n-zu-n“ oder „viele-zu-viele“ Eine Ausprägung der einen Entität kann keiner, einer oder vielen Ausprägungen der anderen Entität zugeordnet werden, und umgekehrt. Wird auch mit n:m oder m:m abgekürzt.	**verheiratet (Polygamie)** Ein Mann ist mit keiner, einer oder vielen Frauen verheiratet. Eine Frau ist mit keinem, einem oder vielen Männern verheiratet. **kauft** Ein Kunde kauft keinen, einen oder viele Artikel. Ein Artikel (der Typ, nicht der konkrete Artikel) wird von keinem, einem oder vielen Kunden gekauft.

Zeitliche Rahmenbedingungen

Beachten Sie, dass alle Informationen in einer relationalen Datenbank wie in einem Entity-Relationship-Modell immer zu einem bestimmten Zeitpunkt gelten. So kann sich die Beziehung „ist-verheiratet“ im Lauf der Zeit durchaus von einer 1:1-Beziehung zu einer n:n-Beziehung ändern. Dies ist aber nur dann relevant, wenn der zeitliche Verlauf in die Datenbank hineinmodelliert werden soll. In diesem Fall ändern sich auch die Schlüssel und viele andere Aspekte. Die Modellierung der Zeit in Form einer Historie stellt besondere Anforderungen an eine Datenbank, die hier nicht weiter vertieft werden sollen.

Kardinalität

Abschließend verdient hier noch das Konzept der Kardinalität Erwähnung, hat es doch unmittelbaren Einfluss auf unser späteres Datenbankschema, insbesondere auf `NULL`-Werte in den Fremdschlüsseln und damit auf die Nutzung von `INNER JOIN` und `OUTER JOIN`.

Definition

Die Kardinalität gibt an, wie viele Ausprägungen einer Entität an einer Beziehung minimal und maximal beteiligt sein können. Es wird dabei das Minimum und das Maximum angegeben. Das Maximum kennen wir schon von den Grundtypen, es ist im Allgemeinen 1 oder n, also maximal einer oder viele. Das Minimum gibt zusätzlich an, wie viele Ausprägungen einer Entität mindestens zu einer Ausprägung der anderen Entität gehören müssen. Es wird also nicht nur das Maximum, sondern auch das Minimum angegeben.

Die gängigen Werte für das Minimum sind:

- 0: keiner
- 1: einer

Beispiele

Die Schreibweise ist:

Entität1 [Minimum,Maximum] Beziehung [Minimum,Maximum] Entität2

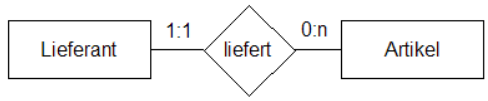
Betrachten wir die folgenden Beziehungen:

```
PERSON1 [0,1] ist verheiratet mit [0,1] PERSON2
```

Eine `PERSON1` kann mit maximal einer `PERSON2` verheiratet sein, muss es aber nicht, und umgekehrt.

```
LIEFERANT [1:1] liefert [0:n] ARTIKEL
```

Abbildung 7.3
Lieferant liefert Artikel.



Ein Lieferant liefert mindestens keinen Artikel und höchstens viele Artikel. Dies macht Sinn, wenn wir auch mögliche Lieferanten mit in die Datenbank aufnehmen, von denen wir noch keine Ware bezogen haben. Die „0“ bedeutet, dass es später Datensätze in der Lieferantentabelle geben kann, die keinen passenden Satz in der Artikeltabelle besitzen. In bestimmten Fällen wird dann ein `OUTER JOIN` notwendig, um beispielsweise festzustellen, welche Lieferanten im Moment noch keine Artikel liefern.

Umgekehrt bedeuten die Kardinalitäten in obigem Beispiel, dass ein Artikel von mindestens einem und höchstens einem Lieferanten geliefert wird. Wir haben also für alle Artikel Lieferanten und beziehen die Ware immer nur von einem bestimmten Lieferanten.

Zuordnung der Kardinalitäten

Sie haben sich vielleicht etwas über die Zuordnung der Kardinalitäten gewundert. Man geht immer von einer Entität aus, geht über die Beziehung und liest dann an dem gegenüberliegenden Eintrag `[Minimum:Maximum]` ab, wie viele Ausprägungen der anderen Entität zu einer Ausprägung unserer

Ausgangsentität kombiniert werden können. Bei der Betrachtung aus Sicht einer Entität wird immer genau eine Ausprägung dieser Entität betrachtet. Die Zuordnungsmöglichkeiten ergeben sich aus den Kardinalitäten der Beziehung auf der Seite der anderen Entität.

Rekursive Beziehungen

Wie im obigen Beispiel „`PERSON ist verheiratet mit PERSON`“ schon gesehen, ist es auch möglich, eine Beziehung zu definieren, die eine Entität mit sich selbst verbindet. Dies nennt man rekursive Beziehungen. Dabei stehen im Allgemeinen nicht dieselben Datensätze miteinander in Verbindung, obwohl auch das möglich ist. Im Fall der „verheiratet“-Beziehung muss man von zwei verschiedenen Datensätzen ausgehen, wer heiratet schon sich selbst?

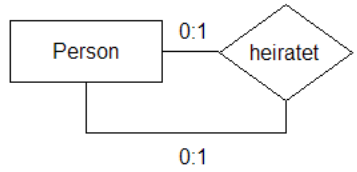


Abbildung 7.4
Rekursive Beziehung

Eine andere häufig auftretende rekursive Beziehung ist die zwischen Bauteilen. Ein Bauteil besteht aus (anderen) Bauteilen. So lassen sich beispielsweise Stücklisten für die Konstruktion oder Produktion speichern. Normalerweise besteht ein Bauteil aus mehreren anderen Bauteilen. Auf der untersten Ebene gibt es dann Bauteile, die aus keinem anderen Bauteil mehr zusammengesetzt werden. Dies erklärt die eine Kardinalität. Umgekehrt wird ein Bauteil beim Zusammenbau in mehreren anderen Bauteilen verwendet und wiederum existieren auf der obersten Ebene Bauteile, die in keinem anderen Bauteil verwendet werden.

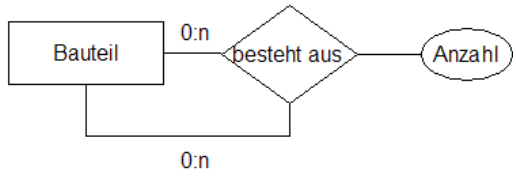


Abbildung 7.5
Bauteile bestehen aus einer Anzahl anderer Bauteile.

Dabei erkennen Sie hier noch die zusätzliche Konstruktion, dass die Beziehung selbst ein Attribut `Anzahl` besitzt. Dies ermöglicht die Modellierung der Situation, dass eine bestimmte Anzahl Bauteile in ein anderes Bauteil eingebaut werden.

7.3 Beispiel BüroFix

Die Entwicklung eines Entity-Relationship-Modells soll an einem konkreten Beispiel nachvollzogen werden. Dazu wird das folgende Beispiel verwendet.

Beispiel BüroFix

Das Unternehmen BüroFix handelt mit einem breiten Sortiment für den Bürobedarf. Es möchte eine Datenbank aufbauen, die ihm Auskunft über Einkauf und Verkauf verschiedener Artikel seines Sortiments gibt. BüroFix besteht aus zwei Firmen, der BüroFix KG und der Büro Versand GmbH. Die BüroFix KG betreibt mehrere Ladengeschäfte, während die Büro Versand GmbH ihre Kunden im Versandhandel beliefert. Zusätzlich hat sie in der Nachbarstadt noch die Paul und Weiss Bürobedarf GmbH übernommen, die dort mehrere Geschäfte betreibt. BüroFix möchte eine gemeinsame Datenbank für die drei Firmen betreiben. Alle Kunden einschließlich der Versandkunden bilden einen gemeinsamen Kundenstamm. Die Waren sind verschiedenen Warengruppen zugeordnet. Jeder Artikel hat einen von mehreren vordefinierten Mehrwertsteuersätzen. Die Kunden erteilen Bestellungen, die aus mehreren Bestellpositionen bestehen können. Jede Bestellposition betrifft einen Artikel. Es sollen Kundenlisten erstellt werden, mit deren Hilfe Kunden gezielt Angebote im Rahmen von Aktionen gemacht werden können. Außerdem will man eine ganze Reihe von Informationen abfragen können, beispielsweise wann welcher Kunde welche Waren gekauft hat, welche Umsätze man mit welchem Artikel macht oder wie hoch die Lagerbestände sind.

Diese Beschreibung soll die Grundlage für die Entwicklung einer kleinen Beispieldatenbank bilden, die schrittweise umgesetzt wird. Die Attribute sind nicht genau definiert und werden im Rahmen der Modellierung sinngemäß ergänzt.

Beispiel Artikel

Die Umsetzung in ein ER-Modell beginnt mit der Suche nach Entitäten. Dazu werden zunächst die Substantive einer genaueren Betrachtung unterzogen. Dazu sind diese im folgenden Text hervorgehoben.

Das Unternehmen BüroFix handelt mit einem breiten Sortiment für den Bürobedarf. Es möchte eine Datenbank aufbauen, die ihm Auskunft über Einkauf und Verkauf verschiedener Artikel seines Sortiments gibt. BüroFix besteht aus zwei Firmen, der BüroFix KG und der Büro Versand GmbH. Die BüroFix KG betreibt mehrere Ladengeschäfte während die Büro Versand GmbH ihre Kunden im Versandhandel beliefert. Zusätzlich hat sie in der Nachbarstadt noch die Paul und Weiss Bürobedarf GmbH übernommen, die dort mehrere Geschäfte betreibt. BüroFix möchte eine gemeinsame Datenbank für die drei Firmen betreiben. Alle Kunden einschließlich der Versandkunden bilden einen gemeinsamen Kundenstamm. Die Waren sind verschiedenen Warengruppen zugeordnet. Jeder Artikel hat einen von mehreren vordefinierten Mehrwertsteuersätzen. Die Kunden erteilen Bestellungen, die aus mehreren Bestellpositionen bestehen können. Jede Bestellposition betrifft einen Artikel. Es sollen Kundenlisten erstellt werden, mit deren Hilfe den Kunden gezielt Angebote im Rahmen von Aktionen gemacht werden

können. Außerdem will man eine ganze Reihe von Informationen abfragen können, beispielsweise wann welcher Kunde welche Waren gekauft hat, welche Umsätze man mit welchem Artikel macht oder wie hoch die Lagerbestände sind.

Eine ganze Reihe der markierten Substantive kommen nicht als Entitäten infrage. So sind „BüroFix“, „BüroFix KG“, „Büro Versand GmbH“ oder „Paul und Weiss Bürobedarf GmbH“ Begriffe, von denen nur eine Instanz existiert. Sie kommen daher als Entität nicht infrage.

Dann gibt es eine Reihe von Begriffen, die als nicht zum Problemumfang zugehörig angesehen werden oder rein technischer Natur sind. Dies sind insbesondere „Bürobedarf“, „Ladengeschäft“, „Versandhandel“, „Nachbarstadt“, „Geschäft“, „Datenbank“ und „Reihe weiterer Informationen“.

Dann gibt es berechenbare Informationen, die später aus der Datenbank gewonnen werden können, aber nicht selbst Bestandteil der Information in der Datenbank sind. In unserem Beispiel sind das „Kundenlisten“, die mit einer SELECT-Anweisung aus den Kunden erstellt werden können. Auch „Angebote im Rahmen von Aktionen“ oder „Umsätze“ deuten auf berechenbare Informationen hin, die aus anderen Informationen gewonnen werden können.

„Lagerbestände“ sind zwar wichtige Informationen, die aber eher als Attribut eines Artikels denn als eigene Entität zu sehen sind. Es handelt sich hier eher um Detailinformationen, die nicht durch weitere Attribute beschrieben werden können oder die einen Schlüssel besitzen.

Somit bleiben zunächst:

- Sortiment
- Kunde
- Artikel
- Firma
- Warengruppe
- Bestellung
- Bestellposition
- Ware
- Mehrwertsteuersatz

Die Liste enthält eine Reihe von zumindest ähnlichen, wenn nicht sogar synonymen Begriffen. Das Sortiment ist letztlich – grob gesprochen – nichts anderes als die Menge Artikel. Ware ist in diesem Zusammenhang als Synonym zu Artikel zu sehen, da es bei der Bestellung der Waren letztlich um die Bestellung von Artikeln geht, kann der Begriff entfallen. Problematisch ist der Mehrwertsteuersatz. Dieser kann als Attribut eines Artikels angesehen werden, da er zunächst nur aus einem einzelnen Wert zu bestehen scheint, der dem Artikel zugeordnet wird. Das Mehrwertsteuersystem beinhaltet aber letztlich drei Gruppen von Artikeln, solchen, die dem vollen Mehrwert-

steuersatz unterliegen, solchen, die dem ermäßigten Steuersatz unterliegen, und solchen, die mehrwertsteuerfrei sind. Die konkreten Prozentsätze sind politischen Änderungen unterworfen. Daher ist hier eine Modellierung denkbar, die eine eigene Entität mit der Art des Mehrwertsteuersatzes (voll, ermäßigt, keiner) als Primärschlüssel und einem zusätzlichen Attribut mit dem konkreten Prozentsatz erlaubt (siehe Abbildung 7.8).

Es bleiben dann letztlich als Kandidaten für Entitäten:

- Kunde
- Artikel
- Firma
- Warengruppe
- Bestellung
- Bestellposition
- Mehrwertsteuersatz

Für das angesprochene Beispiel der BüroFix kann die Modellierung der Entität „Kunde“ wie in Abbildung 7.6 geschehen. Neben der Entität selbst im Zentrum der Darstellung sind alle Attribute um die Entität gruppiert. Als Primärschlüssel ist die Kunden-ID durch Unterstreichung markiert. Wichtig ist die Wahl dieses Schlüssels. Hätten bei der BüroFix die einzelnen Teilunternehmen eigene Kundenstammdaten und würden sie eigene Kundennummern vergeben, so würde die Kunden-ID nur im Zusammenhang mit der Mandanten-ID einen Primärschlüssel ergeben.

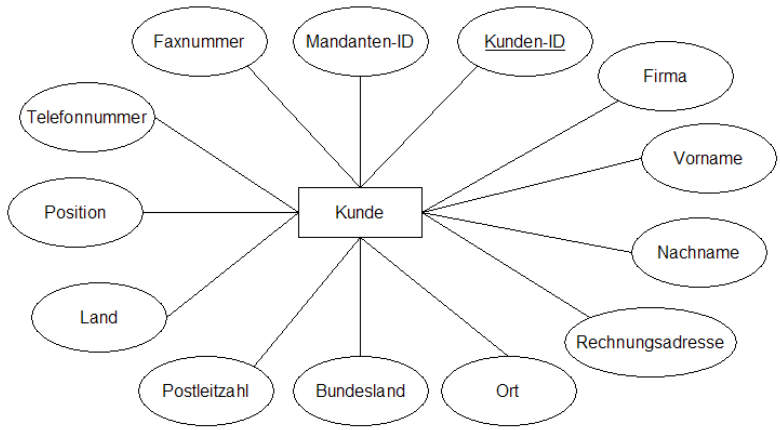


Abbildung 7.6
Entität „Kunde“

Die Auswahl der Attribute für die Entität „Kunde“ ergibt sich hier nicht nur aus der obigen Beschreibung, sondern auch aus einer Analyse des inhaltlichen Umfeldes. Ein beschreibender Text – zumal ein so kurzer Text wie oben – kann nie alle notwendigen Attribute beinhalten. In der Praxis sind daher stets zusätzliches Expertenwissen und ein gezielter Informationssammlungsprozess notwendig.

Ähnlich sieht es mit den anderen Entitäten aus. So zeigt Abbildung 7.7 die Modellierung der Entität „Artikel“. Die Begrifflichkeiten sind wiederum der Anwendungssituation entnommen. Der Listenpreis ist als Nettopreis zu verstehen. In der Realität ist dann die Mehrwertsteuer zu addieren.

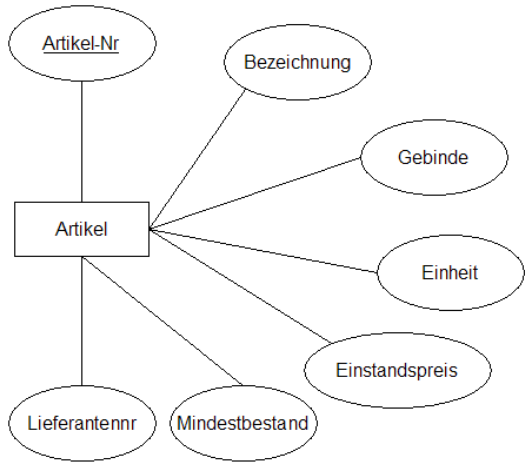


Abbildung 7.7
Entität „Artikel“

Die Höhe des Mehrwertsteuersatzes richtet sich nach der Art des Artikels. So sind die meisten Lebensmittel mit dem halben Mehrwertsteuersatz belastet, während die Büroartikel mit dem vollen Satz zu versteuern sind. Es sind aber nicht nur unterschiedliche Mehrwertsteuersätze zu berücksichtigen, der Mehrwertsteuersatz ändert sich auch unter Umständen durch politische Entscheidungen im Zeitverlauf. Derartigen Situationen kann dadurch Rechnung getragen werden, dass der aktuelle Wert, hier der tatsächliche Mehrwertsteuersatz, nicht direkt bei dem Artikel gespeichert wird. Stattdessen wird eine eigene Entität für die Mehrwertsteuer dargestellt, in der die Art des Mehrwertsteuersatzes (voll, ermäßigt, kein) und die tatsächliche Höhe des Prozentsatzes gespeichert wird (siehe Abbildung 7.8). Ändert sich der Prozentsatz, wird er hier geändert. Anschließend müssen die Entitäten natürlich noch miteinander verbunden werden.

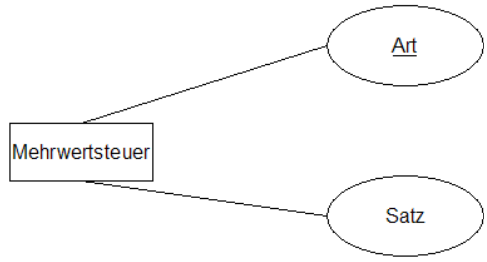


Abbildung 7.8
Entität „Mehrwertsteuer“

Sind die Entitäten festgelegt, müssen die Beziehungen zwischen den Entitäten ermittelt und beschrieben werden.

Sind die Entitäten eher den Substantiven einer textuellen Problembeschreibung zu entnehmen, entsprechen die Verben eher den Beziehungen. Gerade die Verben, die sich auf Substantive beziehen, die bereits als Entitäten akzeptiert wurden, können sinnvolle Hinweise auf Beziehungen liefern. Dazu wollen wir noch einmal den ursprünglichen Text analysieren.

Das Unternehmen BüroFix handelt mit einem breiten Sortiment für den Bürobedarf. Es möchte eine Datenbank aufbauen, die ihm Auskunft über Einkauf und Verkauf verschiedener Artikel seines Sortiments gibt. BüroFix besteht aus zwei Firmen, der BüroFix KG und der Büro Versand GmbH. Die BüroFix KG betreibt mehrere Ladengeschäfte, während die Büro Versand GmbH ihre Kunden im Versandhandel beliefert. Zusätzlich hat sie in der Nachbarstadt noch die Paul und Weiss Bürobedarf GmbH übernommen, die dort mehrere Geschäfte betreibt. BüroFix möchte eine gemeinsame Datenbank für die drei Firmen betreiben. Alle Kunden einschließlich der Versandkunden bilden einen gemeinsamen Kundenstamm. Die Waren sind verschiedenen Warengruppen zugeordnet. Jeder Artikel hat einen von mehreren vordefinierten Mehrwertsteuersätzen. Die Kunden erteilen Bestellungen, die aus mehreren Bestellpositionen bestehen können. Jede Bestellposition betrifft einen Artikel. Es sollen Kundenlisten erstellt werden, mit deren Hilfe den Kunden gezielt Angebote im Rahmen von Aktionen gemacht werden können. Außerdem will man eine ganze Reihe von Informationen abfragen können, beispielsweise wann welcher Kunde welche Waren gekauft hat, welche Umsätze man mit welchem Artikel macht oder wie hoch die Lagerbestände sind.

Problembeschreibung aus Beziehungssicht

Das Unternehmen BüroFix handelt mit einem breiten Sortiment für den Bürobedarf. Es möchte eine Datenbank aufbauen, die ihm Auskunft über Einkauf und Verkauf verschiedener Artikel seines Sortiments gibt. BüroFix besteht aus zwei Firmen, der BüroFix KG und der Büro Versand GmbH. Die BüroFix KG betreibt mehrere Ladengeschäfte, während die Büro Versand GmbH ihre Kunden im Versandhandel beliefert. Zusätzlich hat sie in der Nachbarstadt noch die Paul und Weiss Bürobedarf GmbH übernommen, die dort mehrere Geschäfte betreibt. BüroFix möchte eine gemeinsame Datenbank für die drei Firmen betreiben. Alle Kunden einschließlich der Versandkunden bilden einen gemeinsamen Kundenstamm. Die Waren sind verschiedenen Warengruppen zugeordnet. Jeder Artikel hat einen von mehreren vordefinierten Mehrwertsteuersätzen. Die Kunden erteilen Bestellungen, die aus mehreren Bestellpositionen bestehen können. Jede Bestellposition betrifft einen Artikel. Es sollen Kundenlisten erstellt werden, mit deren Hilfe den Kunden gezielt Angebote im Rahmen von Aktionen gemacht werden können. Außerdem will man eine ganze Reihe von Informationen abfragen können, beispielsweise wann welcher Kunde welche Waren gekauft hat, welche Umsätze man mit welchem Artikel macht oder wie hoch die Lagerbestände sind.

Im Text sind jetzt gezielt Verben hervorgehoben, die im Zusammenhang mit den ausgewählten Entitäten stehen. Setzen Sie die Beziehungen in das ERM um, so erhalten Sie das Bild aus Abbildung 7.9.

Sie sehen, dass ein Kunde nicht zwingend eine Bestellung abgegeben haben muss, es werden also auch Interessenten gespeichert. Umgekehrt kann eine Bestellung nur von genau einem Kunden aufgegeben werden. Eine Bestellung besteht aus mindestens einer Bestellposition, kann aber natürlich beliebig viele Positionen umfassen. Die Bestellposition dient der Speicherung der Bestellmenge für einen Artikel. Dementsprechend wird einer Bestellposition auch genau ein Artikel zugeordnet. Der Artikel beschreibt den Typ, also nicht das einzelne Exemplar. Entsprechend kann ein Artikel mehreren Bestellpositionen zugeordnet werden. Ist ein Artikel noch nie von einem Kunden bestellt worden, ist er bisher auch keiner Bestellung und damit keiner Bestellposition zugeordnet. Entsprechend ist hier auch das Minimum 0 zugelassen. Jeder Artikel gehört zu genau einer Warengruppe, die ihrerseits aus vielen, aber mindestens einem Artikel besteht. Jedem Artikel wird ein Mehrwertsteuersatz eindeutig zugeordnet. Jeder Mehrwertsteuersatz kann für beliebig viele Artikel gelten. Es kann auch geschehen, dass im Sortiment für einen Mehrwertsteuersatz kein Artikel geführt wird.

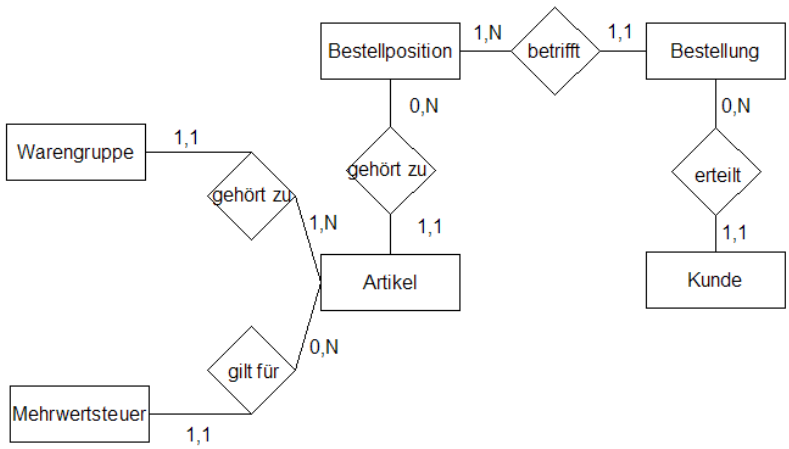


Abbildung 7.9 Komplettes Entity-Relationship-Modell ohne Attribute

7.4 Umsetzung in das relationale Modell

Wir haben im Zusammenhang mit SQL immer vom relationalen Modell gesprochen. Jetzt ist das Entity-Relationship-Modell hinzugekommen. Sein Vorteil ist, dass es einen guten Überblick in der Entwurfsphase gibt. Problematisch ist nur, dass es damit zwei Modelle zur Beschreibung der Informationen gibt.

Das relationale Modell verwendet eine etwas andere Terminologie als das ER-Modell. Glücklicherweise sind das ER-Modell und das relationale Modell aber relativ gut verträglich. Daher lässt sich ein ER-Modell relativ einfach in ein relationales Modell umformen, das dann unmittelbar in eine relationale Datenbank wie beispielsweise MySQL, MS Access oder Oracle umgesetzt werden kann.

Die wichtigsten Regeln für die Umsetzung sind im Folgenden aufgelistet.

Regeln für Entitäten

Regeln für die Transformation von Entitäten

- Jede Entität wird zu einer Tabelle.
- Alle Attribute einer Entität werden als Felder in die Tabelle übernommen.
- Die Primärschlüssel bleiben erhalten und werden ebenfalls Primärschlüssel.
- Die Namen werden soweit möglich übernommen, sonst gemäß den SQL-Regeln umgeformt.

Regeln für Beziehungen

Regeln für die Transformation von Beziehungstypen

Bei der Umsetzung der Beziehungen ist in Abhängigkeit der verwendeten Kardinalitäten ein unterschiedliches Vorgehen notwendig. Insbesondere die n:n-Beziehungen erfordern dabei besondere Aufmerksamkeit.

Standardfall (1:1 und 1:n)

Der erste Fall betrifft alle Beziehungstypen, die binär sind, also nur zwei Entitäten beinhalten, 1:1 oder 1:n als Kardinalität haben und ohne eigene Attribute sind. Dies sollten normalerweise etwa 70–80 % aller Beziehungen sein. In diesen Fällen muss nur die Primär-/Fremdschlüsselbeziehung vorbereitet werden:

- Für jede 1:n-Beziehung wird der Primärschlüssel der Tabelle auf der „n-Seite“ als Fremdschlüssel in die Tabelle der „1-Seite“ übernommen.
- Für jede 1:1-Beziehung wird auf einer Seite der Primärschlüssel der anderen Tabelle als Fremdschlüssel übernommen. Dabei wird der Fremdschlüssel für **die** Seite gewählt, die tendenziell die „abhängigere“ Entität darstellt.

Spezialfall n:n-Beziehungen und Beziehungen mit eigenem Attribut

Diese Beziehungen sollten die weiteren 15–25 % ausmachen.

- Für jede n:n-Beziehung wird eine eigene Tabelle erstellt. Diese enthält die Primärschlüssel der beiden beteiligten Entitäten als Fremdschlüssel. Alle Attribute der Beziehung bilden den Primärschlüssel.
- Jede Beziehung mit Attributen wird in eine eigene Tabelle umgesetzt. Die Attribute werden übernommen. Zusätzlich werden die Primärschlüssel der beiden beteiligten Entitäten als Fremdschlüssel übernommen. Der Primärschlüssel besteht aus allen Attributen einschließlich der Fremdschlüsselattribute (gegebenenfalls Prüfung, ob weniger ausreichend sind). Der Name wird aus dem Namen des Beziehungstyps und/oder den Entitätstypen gebildet.

Spezialfall Beziehungen mit drei oder mehr Entitäten

Dies sind die verbleibenden vielleicht 5 % aller Beziehungen.

- Jede Beziehung mit einem Grad größer als 2, also mit mehr als zwei beteiligten Entitäten, wird in eine eigene Tabelle umgesetzt. Die Primärschlüssel aller beteiligten Entitäten werden als Fremdschlüssel übernommen. Alle zusammen bilden gemeinsam mit eventuell noch vorhandenen weiteren Attributen den Primärschlüssel. Es sollte hier besonders sorgfältig geprüft werden, ob ein Teil der Attribute entfallen kann.

Regeln für Attribute

Die Attribute werden ebenfalls komplett übernommen und in die Tabellen integriert. Dabei sind erstmals auch die physischen Datentypen zu vergeben, also etwa `CHAR()`, `VARCHAR()`, `INTEGER`, `FLOAT`, `BOOLEAN`, `DATE`, `TIME`, `TIMESTAMP` oder ein anderer Datentyp der Datenbank.

Beispiel

Für das Beispiel ergeben sich im relationalen Modell folgende Tabellen (Felder sind bereits an Namenskonventionen angepasst und teilweise reduziert):

Die Mehrwertsteuertabelle besteht aus zwei Feldern, der Art des Mehrwertsteuersatzes und dem tatsächlichen Mehrwertsteuersatz `mwst_satz` als Angabe in Prozent.

```
mwst (mwst_art ENUM('voll','ermässigt','kein'), mwst_satz INT)
```

Ein typischer Datensatz wäre hier:

```
voll;19
```

Die Kundentabelle besitzt die Mandantennummer `mid` und die Kunden-ID `kid` als gemeinsamen Primärschlüssel, weil die Kunden nicht über die Mandanten (die drei Firmen) eindeutig sind.

```
kunden (mid INT, kid VARCHAR(50), firma VARCHAR(50), anrede VARCHAR
(10), nachname VARCHAR(50), vorname VARCHAR(30), rechnungsadresse VAR-
CHAR(255), strasse VARCHAR(50), PLZ CHAR(5), ort VARCHAR(50), geburts-
datum DATE)
```

Ein typischer Datensatz wäre hier:

```
1;1;;Frau;Sibum;Jutta;;Hannoversche Straße 6;29223;Celle;02.04.65
```

Warengruppen sollen die einzelnen Artikel des Sortiments gruppieren. Die Warengruppe dient der Speicherung der Nummer und der Bezeichnung aller Warengruppen.

```
warengruppe (kennziffer INT, warengruppe VARCHAR(255))
```

Ein typischer Datensatz wäre hier:

```
1;Papier- und Papierprodukte
```

Der Artikel beschreibt das einzelne Angebot. Die Preise sind als `DECIMAL` realisiert, um eine stellengenaue Speicherung zu ermöglichen. Hier werden auch Beziehungen durch Fremdschlüsselfelder realisiert. Das Feld `wgknz` stellt die Verbindung zur Warengruppe her. Mit dem Feld `mwst_art` wird die Verbindung zur Mehrwertsteuer realisiert.

```
artikel (anr INT, bezeichnung VARCHAR(255), gebinde INT, einheit VAR-
CHAR(30), wgknz INT, einstandspreis DECIMAL(6,2), listenpreis DECI-
MAL(6,2), mwst_art ENUM('voll','ermässigt','kein'), mindestbestand
INT)
```

Ein typischer Datensatz wäre hier:

```
1002;Universalpapier Standard;500;Blatt;1;2.48;5.99;voll;2000
```

Die Bestellung beinhaltet neben dem Bestelldatum vor allem mit der `mid` und der `kid` die Fremdschlüssel für die Beziehung zum bestellenden Kunden.

```
bestellung (mid INT, bnr INT, kid VARCHAR(50), bestelldatum DATE);
```

Ein typischer Datensatz wäre hier:

```
1;1;4711;15.10.2007
```

Die Bestellposition beschreibt die Menge, die von einem Artikel innerhalb einer bestimmten Bestellung bestellt wird. Über die `mid` und die `bnr` wird die Beziehung zur Bestellung hergestellt. Die Beziehung zum bestellten Artikel wiederum wird über die `anr` hergestellt.

```
bestell_position (mid VARCHAR(50), bnr INT, pos INT, anr INT, anzahl
INT)
```

Ein typischer Datensatz wäre hier:

```
1;8876;1;10;2
```

7.5 Sinn und Unsinn der Normalisierung

7.5.1 Redundanz und Anomalien

Sie haben wahrscheinlich bereits bei der Erstellung der einfachen `SELECT`-Anweisungen in Kapitel 4 festgestellt, dass die Beziehungen immer wieder Probleme verursachen können. Wenn wir uns in Kapitel 9 mit Unterabfragen beschäftigen, wird die Sache noch unangenehmer. Es stellt sich die Frage, warum man sich das denn überhaupt antut und nicht alle Informationen in eine Tabelle legt. „Ein paar Felder mehr und dann jeweils die richtigen verwenden“ müsste doch auch gehen. Tatsächlich hat es sogenannte Datenbankmanagementsysteme gegeben, die diesen Ansatz verfolgt haben, das „berühmt-berüchtigste“ ist *dBase*. dBase ist kein relationales Datenbanksystem, sondern eine große Tabelle, man könnte eigentlich auch gleich Excel nehmen (was einige Anwender auch tun, wie meine Erfahrungen in der Praxis leider gezeigt haben).

Redundanz

Das grundsätzliche Problem heißt dann Redundanz, also die mehrfache Abspeicherung desselben Sachverhaltes. Redundanz ist speicherintensiv und fehleranfällig.

Mit dem Speicherplatz könnte man bei kleinen Datenbanken noch leben (und bei einigen größeren – wie im Data Warehouse-Bereich – ist sie zum Zweck der schnelleren Abfrage sogar gewollt), aber die Fehleranfälligkeit stellt ein großes Problem dar.

Betrachten Sie die Beispieltabelle `Bestellungen` unserer Bürofix-Datenbank in Abbildung 7.10. Die vollständige Tabelle ist als Excel-Tabelle unter *Datenbanken/MSAccess/Artikel* unter *Bestellungen_Kap7_redundant.xls* auf der CD enthalten. Sie sehen, dass die Information über den Mitarbeiter 777

```
777, Olsen, Edwin
```

allein in dem kleinen Ausschnitt siebenmal auftaucht. Es reicht vollkommen, wenn Sie diesen Zusammenhang einmal speichern, nicht siebenmal. Dann genügt die Mitarbeiternummer 777, um eine Bestellung aufzunehmen, und ein `JOIN` holt jederzeit den Namen des Mitarbeiters dazu.

Mitarbeiternr	Mitarbeiternai	Mitarbeitervor	PLZ	Ort	Kundennr	Kundenname	Kunden_PLZ	Kunden_Ort
777	Olsen	Edwin	29225	Celle	2	EWALD Kuhn KG	29223	Celle Windmühlenstr. 12
777	Olsen	Edwin	29225	Celle	2	EWALD Kuhn KG	29223	Celle Windmühlenstr. 12
777	Olsen	Edwin	29225	Celle	2	EWALD Kuhn KG	29223	Celle Windmühlenstr. 12
277	Wichtig	Willi	30156	Hannover	3	Hans-Joachim Bit	29320	Hermannsburg Lange Straße 1
277	Wichtig	Willi	30156	Hannover	3	Hans-Joachim Bit	29320	Hermannsburg Lange Straße 1
777	Olsen	Edwin	29225	Celle	6	Hobelix OHG	30230	Hannover Lange Laube 2
777	Olsen	Edwin	29225	Celle	6	Hobelix OHG	30230	Hannover Lange Laube 3
777	Olsen	Edwin	29225	Celle	17	EWALD Kuhn KG	29223	Celle Windmühlenstraße 12
777	Olsen	Edwin	29225	Celle	17	EWALD Kuhn KG	29223	Celle Windmühlenstr. 15
277	Wichtig	Willi	30156	Hannover	25	Clean GmbH	29225	Celle Lauensteinplatz 8
277	Wichtig	Willi	30156	Hannover	25	Clean GmbH	29225	Celle Lauensteinplatz 8
777	Wichtig	Willi	30156	Hannover	25	Clean GmbH	29225	Celle Lauensteinplatz 9

Abbildung 7.10
Ausschnitt einer redundanten Tabelle

Betrachten Sie den Kunden „CLEAN GmbH“. In Abbildung 7.10 sehen Sie wieder die Redundanz. Hier besteht das Problem darin, dass zu jedem Auftrag immer wieder alle Daten eines Kunden abgelegt werden. Die Daten des Kunden „Clean GmbH“, hier seine Kundennummer, die PLZ und weitere Daten wie Ort, Straße und Hausnummer, werden für jeden Auftrag dieses Kunden immer wieder abgelegt. Das ist nicht nur speicherintensiv, sondern auch fehleranfällig. Woher wissen wir eigentlich, dass es Lauensteinplatz 8 und nicht Lauensteinplatz 9 ist, wie auch einmal angegeben. Welche Angabe stimmt?

Anomalien

Derartige Fehler sind sogenannte *Anomalien*. Anomalien sind widersprüchliche Informationen in einer Datenbank.

Anomalien entstehen durch Redundanz. Wird dieselbe Information mehrfach – redundant – gespeichert, kann es zu Fehlern kommen, die dann Anomalien darstellen. Jeder, der in der Praxis Kundendatenbanken gesehen hat, weiß, warum wir Kataloge mehrmals bekommen, unsere Nachmieter noch nach Monaten unsere Post erhalten und ehemalige Adressen scheinbar unauslöschlich sind. Hier werden Informationen redundant abgelegt und bei der teilweisen Änderung entstehen Anomalien.

Einfügeanomalie (INSERT)

Anomalien können bei der Eingabe entstehen. So können bei der erneuten Eingabe der Kundendaten Fehler gemacht werden. Wird beispielsweise bei dem nächsten Auftrag der Kundenname des Kunden 2 versehentlich als „Ewal Kuhn KG“ statt „Ewald Kuhn KG“ eingegeben, so gibt es in der Datenbank in der Bestelltabelle Datensätze, die die Information beinhalten, dass der Kunde 2 „Ewald KUHN KG“ heißt, andere Datensätze geben die Information wieder, dass er „Ewal Kuhn KG“ heißt. Das ist eine Einfügeanomalie, da der Fehler bei der Neueingabe eines Datensatzes (`INSERT`-Anweisung) auftritt.

Änderungsanomalie (UPDATE)

Sollte der Kunde umziehen, müssen alle Aufträge nachträglich geändert werden. Zieht unser Kunde „Ewald Kuhn KG“ um und ist seine neue Postleitzahl 29225 statt 29223, so ist dies nicht nur an einer Stelle, sondern in allen Datensätzen der Bestelltabelle, die Bestellungen dieses Kunden enthalten – und wer weiß wo sonst noch – zu ändern. Dies ist nicht nur aufwendig, sondern auch fehleranfällig. Werden Aufträge vergessen, so haben wir wieder einen Widerspruch – eine Änderungsanomalie (`UPDATE`-Anweisung).

Löschanomalie (DELETE)

Vermissen Sie noch eine SQL-Anweisung aus Kapitel 6? Richtig, auch die `DELETE`-Anweisung kann in nicht redundanzfreien Datenbanken Probleme verursachen. Stellen Sie sich vor, Sie wollen die Daten über den Kunden „Ewald Kuhn KG“ löschen, weil dieser den Geschäftsbetrieb eingestellt hat. Wollen Sie dann alle seine Aufträge löschen? Dies könnte problematisch sein, wenn Sie später Ihren Umsatz insgesamt bestimmen wollen. Andererseits haben Sie Aufträge ohne Kunden, wenn Sie die Aufträge in der Datenbank lassen und nur den Kunden löschen wollen. Wenn Sie dann beispielsweise die Kundennummer noch als Fremdschlüssel für Ihren `INNER JOIN` definiert haben, müssen Sie jetzt einen künstlichen „Dummy“ erfinden.

Auch umgekehrt können Probleme entstehen. Stellen Sie sich vor, Sie wollen alle Aufträge, die älter als 2 Jahre sind, aus Ihrer Datenbank entfernen. Hat Ihr Kunde „Ewald Kuhn KG“ in den letzten 2 Jahren nichts bestellt, sind dann hinterher auch seine Kundennummer, seine Adresse und andere Angaben verschwunden. Schade eigentlich für Ihr Marketing und Ihren Vertrieb.

Tipp

Redundanz kann zu Anomalien in Ihrer Datenbank führen. Vermeiden Sie Redundanz durch Normalisierung, wenn dies möglich ist. Dies ist der Sinn der Normalisierung. Beachten Sie aber auch den Hinweis zum Unsinn der Normalisierung.

7.5.2 Normalisierungsziele

Redundanz ist also ein Problem. Redundanz entsteht immer dann, wenn zu viele Informationen in eine Tabelle aufgenommen werden. Immer dann, wenn Felder noch von einem oder mehreren anderen Feldern in derselben Tabelle außer dem Primärschlüssel abhängig sind, gehören sie nicht in diese Tabelle. Wenn ich die Mitarbeiternummer kenne, kann ich die restlichen Daten des Mitarbeiters in einer anderen Tabelle nachsehen. Dort brauche ich sie nur einmal abzulegen. Immer dann, wenn ich eine Artikelnummer kenne, „joine“ ich die Artikeltabelle, die die benötigten Informationen einmal für jeden Artikel enthält, oder die Mitarbeitertabelle für die Mitarbeiternamen oder die Kundentabelle für die Kundendaten.

Redundanzfreiheit zu erzeugen, bedeutet daher immer Tabellen so zu zerlegen, dass jede Information nur einmal abgespeichert wird und bei Bedarf die Informationen in der `SELECT`-Anweisung wieder zusammengeführt werden können.

Wir wissen jetzt also, dass es sinnvoll ist, Tabellen zu zerlegen und über Beziehungen zu verbinden, anstatt alle Informationen in einer Tabelle zu speichern. Bleibt noch das Problem, wie man diese Zerlegung sinnvoll gestaltet, damit zusammengehörige Informationen zusammenbleiben, andererseits aber auch Redundanz vermieden werden kann.

Normalisierungsverfahren

Der Weg zu diesem Ziel ist ein Verfahren, das als Normalisierung bezeichnet wird.

Das Ziel der Normalisierung ist die

- Verhinderung der redundanten Speicherung identischer Sachverhalte und
- Vermeidung von Anomalien beim Einfügen, Ändern und Löschen von Datensätzen.

Sie kennen das folgende Beispiel von oben. In der Tabelle `Bestellung` besteht das Problem darin, dass zu jedem Auftrag immer wieder alle Daten eines Mitarbeiters und eines Kunden gespeichert werden. Das sehen Sie bereits an der ersten, zweiten und dritten Zeile. Die Daten des Kunden „Ewald Kuhn KG“, hier seine Kundennummer, die PLZ und der Ort, werden für jeden Auftrag dieses Kunden immer wieder gespeichert. Dies ist Redundanz.

Die Lösung bestünde hier in der Aufspaltung in mehrere Tabellen. Eine Tabelle enthält die Kundendaten, einmalig für jeden Kunden. Die andere Tabelle enthält die Aufträge und lediglich einen Fremdschlüssel (die `Kundennr`), um die Beziehung zu dem passenden Datensatz in der Tabelle `Kunde` herstellen zu können. Die dritte Tabelle enthält die Mitarbeiterdaten.

Dieser Vorgang der Aufspaltung von Tabellen nach bestimmten Kriterien, die Normalisierung, kann in einem klar strukturierten Verfahren erfolgen. Das Verfahren beruht auf einem simplen mathematischen Prinzip, der funktionalen Abhängigkeit.

7.5.3 Funktionale Abhängigkeit

Stellen Sie sich vor, Sie stehen auf einem Turm und lassen einen Stein fallen. Sie sind sicher, dass unten niemand steht, und können zu bestimmten Zeitpunkten die Strecke messen, die der Stein zurückgelegt hat.

Sie kommen zu den Ergebnissen in der Tabelle 7.3.

t in Sekunden	0	1	2	3
s in Metern	0	4,905	19,62	44,145
Die Formel dazu lautet übrigens: $s = 4{,}905\ t^2$				

Tabelle 7.3
Funktionale Abhängigkeit zwischen Zeit und Weg

Auf diesem Weg hat Galileo Galilei übrigens am Schiefen Turm von Pisa die Erdbeschleunigung bestimmt, die das Doppelte des oben angegebenen Wertes 9,81 m/s^2 beträgt, aber das sollten Sie dann doch lieber in einer Formelsammlung zur Physik vertiefen. Wichtig für uns ist die folgende Tatsache:

Wenn Sie einen Wert für `t` kennen, können Sie den Wert für `s` eindeutig bestimmen. Das nennt man *funktionale Abhängigkeit*.

Geschrieben wird die funktionale Abhängigkeit als Pfeil `->`, der bedeutet „daraus ergibt sich". In obigem Beispiel könnten wir also schreiben:

```
0 -> 0
1 -> 4,905
2 -> 19,62
3 -> 44,145
```

oder allgemein

```
t -> s
```

In der Sprache der relationalen Datenbanken heißt das: Aus den Werten des Datenfeldes `t` kann in jedem Datensatz eindeutig der Wert des Datenfeldes `s` bestimmt werden. Anders ausgedrückt: `s` ist von `t` funktional abhängig.

Beispiele:

- **(Kundennummer -> Kundenname)** – Der Kundenname ist funktional von der Kundennummer abhängig. Wenn die Kundennummer bekannt ist, ist der Kundenname eindeutig ermittelbar. Wenn die Nummer „2" ist, ist der Name „Ewald Kuhn KG". In einer Tabelle für Bestellungen muss natürlich die Kundennummer gespeichert werden. Es wäre aber falsch, dort auch den Kundennamen zu speichern, dieser sollte vielmehr aus einer Kundentabelle mit der Kundennummer ermittelt werden.
- **(PID -> Geburtsdatum)** – Das Geburtsdatum ist von der Personen-Identifikationsnummer funktional abhängig. Ist die PID „5" bekannt, kann daraus das Geburtsdatum „5.10.1962" eindeutig bestimmt werden. Das Geburtsdatum gehört aber in die Personentabelle, da es zu einer Person gehört. Die funktionale Abhängigkeit von der PID ist gewollt und muss sogar gelten, schließlich sollen mithilfe der PID als Primärschlüssel der gesamte Datensatz und damit auch alle Werte im Datensatz ermittelt werden können.
- **(Ort, Straße, Hausnummer -> PLZ)** – Sind Ort, Straße und Hausnummer bekannt, kann in Deutschland die Postleitzahl eindeutig bestimmt werden. Sie sehen, es können auch mehrere Datenfelder beteiligt sein. Hier ist die Lage allerdings schwierig. Die Alternative zur Speicherung der PLZ wäre eine Tabelle mit allen beteiligten Feldern und einem komplizierten Schlüssel. Hier wäre eine Abwägung zwischen den Nachteilen der Redundanz und dem Vorteil der schnellen Verfügbarkeit notwendig.

Zusammenfassung

Eine funktionale Abhängigkeit eines Datenfeldes vom Primärschlüssel ist gut und gewollt, dies ist der Zweck des Primärschlüssels. Funktionale Abhängigkeiten zwischen anderen Datenfeldern deuten immer auf Redundanz hin.

7.5.4 Normalformen

Die Normalisierung ist also das Zerlegen von Tabellen entsprechend den funktionalen Abhängigkeiten. Dies geschieht stufenweise. Mit jeder Stufe wird die mögliche Redundanz verringert. Je nach Art und Umfang der angewendeten Regeln erreichen Sie Tabellen, die sich in einer sogenannten Normalform befinden.

Die bekanntesten Normalformen sind:

- 1NF (1. Normalform)
- 2NF (2. Normalform)
- 3NF (3. Normalform)

Weitere Formen, die aber keine so zentrale praktische Bedeutung haben:

- BCNF (Boyce-Codd-Normalform)
- 4NF (4. Normalform)
- 5NF (5. Normalform)

Die Normalformen sind hierarchisch aufeinander aufbauend. Dies bedeutet, dass eine Tabelle, die in 3NF ist, automatisch auch in 1NF und in 2NF ist. Ziel ist es zumeist, für alle Tabellen die 3NF zu erreichen.

1. Normalform (1NF) „Flat table“

Eine Relation ist in 1NF, wenn 1NF

- kein Datenfeld mehr als einen Wert je Datensatz haben kann und
- Datenfelder vom Primärschlüssel funktional abhängig sind.

Ein Datenfeld, das die Werte der Kundennummer und des Kundennamens enthält, etwa „2 Ewald Kuhn KG“ widerspricht der Grundidee eines relationalen DBMS. Das Feld `Kunden_Ort` ist in diesem Sinne problematisch. Die zweite Bedingung greift das Prinzip der funktionalen Abhängigkeit auf und zeigt, dass funktionale Abhängigkeiten von einem Primärschlüssel gut und gewollt sind.

2. Normalform (2NF) „Minimaler Primärschlüssel“

Eine Relation ist in 2NF, wenn sie in 1NF ist und 2NF

- jedes Datenfeld, das selbst nicht Bestandteil des Primärschlüssels ist, vom ganzen Primärschlüssel (und nicht nur einem Teil) funktional abhängig ist.

Diese zusätzliche Anforderung kann also nur zum Tragen kommen, wenn mindestens zwei Primärschlüsselfelder existieren.

3. Normalform (3NF) „Keine transitiven Abhängigkeiten“

3NF

Eine Relation ist in 3NF, wenn sie in 2NF ist und

- keine funktionalen Abhängigkeiten zwischen den Datenfeldern bestehen, die nicht zum Primärschlüssel gehören.

Die Lösung bei Widersprüchen zu einer der Normalformen ist immer dieselbe:

1. Bei einer Abhängigkeit `a1 -> a2` werden `a1` und `a2` in eine eigene zusätzliche Tabelle „ausgelagert“. In dieser Tabelle wird `a1` der neue Primärschlüssel.
2. Die neue Tabelle wird über einen Fremdschlüssel `a1` in der ursprünglichen Tabelle, der dem Primärschlüssel `a1` der neuen Tabelle entspricht, verbunden.

7.5.5 Grenzen der Normalisierung

Vielleicht kennen Sie den alten Satz schon: „Wenn man einen Hammer hat, sieht jedes Problem wie ein Nagel aus.“ In unserem Fall also: „Wenn man die Normalisierung beherrscht, will man jede Datenbank normalisieren.“

Dies ist nur bedingt sinnvoll, in vielen Fällen nicht mehr notwendig und manchmal sogar kontraproduktiv.

Normalisierung durch Modellierung

Hat man eine Datenbank wie wir hier über ein Entity-Relationship-Modell entwickelt, so ist das Ergebnis in vielen Fällen bereits normalisiert, weil man bereits während der Entwicklung des ERM darauf geachtet hat, nur wirklich zusammenhängende Informationen in einer Entität zusammenzufassen. Es ergibt sich auf „natürliche Weise“ ein normalisiertes Modell. Die Normalisierung ist für die Umstellung bestehender Datenbestände, wie beispielsweise die angesprochene Tabelle *Bestellung_Kap7_redundant.xls,* in relationale Strukturen sinnvoll, bei der Neuentwicklung mithilfe des ERM aber zumeist kaum notwendig.

Daher muss hier das Beispielmodell auch nicht weiter normalisiert werden.

Performance

Der Zweck der Normalisierung ist, wie gesagt, die Vermeidung von Redundanz und Anomalien. Der Preis, den man dafür zahlt, ist eine starke Zersplitterung der Datenbank in viele Tabellen und Beziehungen. Diese Tabellen müssen in einer Abfrage mit einer SELECT-Anweisung wieder gejoint werden. Das ist nicht nur kompliziert, sondern führt auch häufig zu Performance-Problemen. Normalisierte Datenbanken können also bei häufigen komplexen Abfragen Probleme im Antwortzeitverhalten bekommen. Hier ist dann zwischen Abfrage und Änderung abzuwägen. Es gibt Datenbanken im Data Warehouse-Umfeld, die praktisch überhaupt keine Änderungen auf Einzelsatzebene kennen, sondern nur Abfragen und Massenloads.

Die Gefahren durch Redundanz sind hier sehr gering. Eine Anomalie entsteht nur durch einen Programmierfehler, da das Laden der Daten über automatisierte Programme erfolgt. Andererseits müssen viele komplexe Abfragen auf

Massendaten ausgeführt werden. Hier stört die Normalisierung, da sie durch die notwendigen JOIN-Verknüpfungen die Performance der Abfragen negativ beeinflusst.

Pflege über Programme

Das Ändern der Datenbank über Programme ist schließlich ein generelles Argument, das die Anforderungen an eine Normalisierung senkt. Sind die Programme gut getestet und erfolgen keine manuellen Eingaben, so ist die Gefahr von Anomalien erheblich gesenkt. Es bleibt dann noch das Argument des geringeren Speicherplatzes, dem die im Allgemeinen schlechtere Performance normalisierter Datenbanken gegenübersteht.

Dies sind alles keine Argumente, die gegen eine normalisierte Datenbankstruktur als solche sprechen, sondern Aspekte, die bei der Gestaltung der Datenbank zusätzlich zu erwägen sind. Eine gezielte und dokumentierte Denormalisierung – also eine teilweise Aufhebung der Normalisierung – ist aber ein probates und vielfach verwendetes Mittel zur gezielten Leistungssteigerung einer Datenbank bei SELECT-Anweisungen. Weitere Mittel bieten die einzelnen Datenbanken dann im Rahmen ihrer physischen Speicherstrukturen, auf die später in Kapitel 14 noch einzugehen ist.

8

Datenbanken erstellen (SQL-DDL)

8.1 Das Datenbankschema erstellen (CREATE SCHEMA)

Im vorangegangenen Kapitel ist eine Datenbankstruktur neu entwickelt worden. Diese existiert bisher nur „auf dem Papier“. Jetzt soll diese Struktur real in ein Datenbankschema umgesetzt werden. Dies geschieht natürlich wiederum mit SQL. Dabei benutzen wir einen bisher nicht betrachteten Teil von SQL, die sogenannte SQL-DDL (Data Definition Language). Im Gegensatz zur SQL-DML (Data Manipulation Language), die die Basis bis Kapitel 6 bildete, ändert die DDL die Datenbank als solches, nicht nur deren Inhalte.

Beispiel

Sie wollen die im letzten Kapitel definierte Artikeldatenbank in einem realen Datenbankmanagementsystem umsetzen. Dazu wollen Sie eine komplett neue Datenbank in Form eines Datenbankschemas anlegen.

Datenbankschema

Die Begriffe Datenbank und Datenbankschema werden oft synonym verwendet. Tatsächlich ist das *Datenbankschema* die Summe aller Strukturinformationen, also aller Tabellen, Schlüssel, Beziehungen, Indizes und anderer Komponenten. Dies sind letztlich die Metadaten, die die Struktur der tatsächlichen Daten wie in einem Katalog beschreiben. Der SQL-Standard sieht vor, dass es als zentrale Struktur einen Katalog gibt, der ein oder mehrere Schemata beinhalten kann.

Von einer *Datenbank* spricht man in der Regel, wenn man die Gesamtheit aller Daten einschließlich der eigentlichen Nutzdaten meint. Leider geht die Begrifflichkeit hier oft durcheinander. Manchmal werden auch Datenbankschemata als die gesamte Datenbank bezeichnet oder das Schema wird Katalog genannt. Sie sollten sich hier jeweils an die in Ihrem Umfeld gebräuchliche Begrifflichkeit halten.

In diesem Buch sprechen wir immer vom Datenbankschema, wenn wir die Struktur meinen. Die Erstellung dieser Struktur erfolgt mithilfe der SQL-DDL. Die erste Anweisung der SQL-DDL haben Sie bereits kennengelernt, es ist die `CREATE SCHEMA`-Anweisung zur Erzeugung eines neuen Datenbankschemas.

CREATE SCHEMA

CREATE SCHEMA *schemaname*

[DEFAULT CHARACTER SET *zeichensatz]*

[Definition von Tabellen, Domänen, Views, Rechten ....];

Oft wird auch `CREATE DATABASE` synonym verwendet. Die `CREATE SCHEMA`/`CREATE DATABASE`-Anweisung wird von den verschiedenen Datenbanksystemen recht unterschiedlich umgesetzt. So ist beispielsweise im SQL-Standard noch eine Autorisierung vorgesehen oder die Definition mehrere Schemata in einer Anweisung. In der Praxis wird allerdings zumeist nur ein Schema mit einer Anweisung erzeugt und die Definition der Tabellen und anderer Strukturen erfolgt mit getrennten SQL-Anweisungen.

8.1.1 MySQL

MySQL-Schema

Sie haben bereits in Kapitel 3 gesehen, wie ein Datenbankschema von der Kommandooberfläche aus angelegt wird. Jetzt wollen wir den „offiziellen" Weg mit SQL gehen. Starten Sie dazu Ihren MySQL Query Browser. Melden Sie sich mit „root" und „masterkey" an. Sie benötigen kein Standardschema. Dieses wollen wir neu anlegen. Nach der Anmeldung geben Sie als SQL-Anweisung

```
CREATE DATABASE artikel;
```

an. Statt des Schlüsselwortes `DATABASE` kann in MySQL synonym – wie bereits erwähnt – auch das Schlüsselwort `SCHEMA` verwendet werden. Testen Sie es mit

```
CREATE SCHEMA artikel2;
```

Um den Erfolg der SQL-Anweisungen zu sehen, müssen Sie in der Oberfläche noch einen Refresh ausführen. Klicken Sie dazu mit der rechten Maustaste in das SCHEMATA-Fenster auf der rechten Seite und wählen Sie die Option AKTUALISIEREN. Sie sollten in etwa die Darstellung wie in Abbildung 8.1 erhalten.

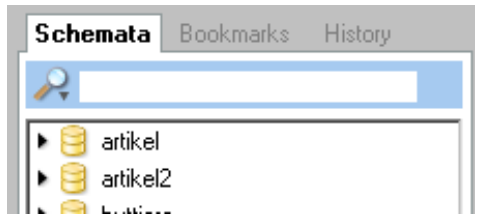


Abbildung 8.1
Zwei neue Datenbankschemata sind erzeugt worden.

Zeichensatz eines Schemas

Ein Schema beinhaltet alle zu einem bestimmten Thema gehörenden Informationen. Man geht daher davon aus, dass diese im Normalfall in einem einheitlichen Zeichensatz abgespeichert werden. MySQL bietet bei der Schema-Definition gerade bei der Wahl des Zeichensatzes und der damit verbundenen Sortierreihenfolge eine große Vielfalt. Die MySQL-Version des `CREATE SCHEMA` ist:

CREATE SCHEMA [IF NOT EXISTS] *schemaname*

[[DEFAULT] CHARACTER SET *zeichensatz]*

[[DEFAULT] COLLATE *sortierreihenfolge];*

Sie können damit für jedes Datenbankschema den zu verwendenden Zeichensatz und die Sortierreihenfolge festlegen. Wird nichts festgelegt, wird der Standard des Servers übernommen.

Mithilfe des Kommandozeilenwerkzeuges *mysql.exe* oder durch direkte Eingabe in den Query Browser können Sie die verfügbaren Zeichensätze ermitteln. Mit

```
SHOW CHARACTER SET;
```

erhalten Sie eine Übersicht über die verfügbaren Zeichensätze (siehe Abbildung 8.2).

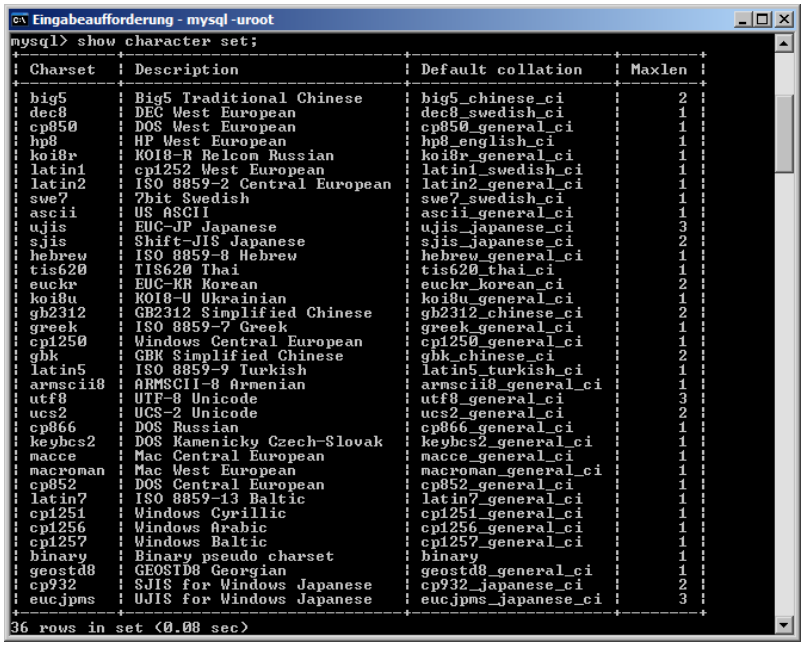

Eingabeaufforderung - mysql -uroot
mysql> show character set;
+----------+-----------------------------+---------------------+--------+
| Charset | Description | Default collation | Maxlen |
+----------+-----------------------------+---------------------+--------+
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
euckr	EUC-KR Korean	euckr_korean_ci	2
koi8u	KOI8-U Ukrainian	koi8u_general_ci	1
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
greek	ISO 8859-7 Greek	greek_general_ci	1
cp1250	Windows Central European	cp1250_general_ci	1
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
armscii8	ARMSCII-8 Armenian	armscii8_general_ci	1
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp866	DOS Russian	cp866_general_ci	1
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci	1
macce	Mac Central European	macce_general_ci	1
macroman	Mac West European	macroman_general_ci	1
cp852	DOS Central European	cp852_general_ci	1
latin7	ISO 8859-13 Baltic	latin7_general_ci	1
cp1251	Windows Cyrillic	cp1251_general_ci	1
cp1256	Windows Arabic	cp1256_general_ci	1
cp1257	Windows Baltic	cp1257_general_ci	1
binary	Binary pseudo charset	binary	1
geostd8	GEOSTD8 Georgian	geostd8_general_ci	1
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3
+----------+-----------------------------+---------------------+--------+
36 rows in set (0.08 sec)

Abbildung 8.2
Zeichensätze in MySQL

Gängige Werte sind hier *latin1*, *ascii* oder *utf8*, wobei für normale Daten mit deutscher Sprache zumeist *latin1* gewählt wird.

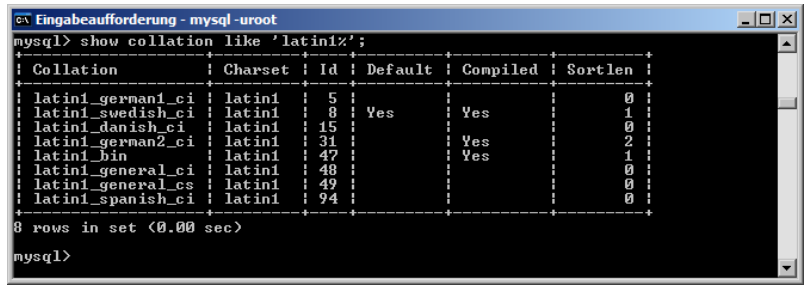
Sortierreihenfolge

Jeder Zeichensatz hat eine oder mehrere Sortierreihenfolgen. So können Sie mit der SQL-Anweisung

```
SHOW COLLATION LIKE 'latin1%';
```

alle Sortierreihenfolgen ermitteln, die für den Zeichensatz *latin1* definiert sind, weil die Namen aller Sortierreihenfolgen stets mit dem Namen des Zeichensatzes beginnen, für den sie definiert sind. Somit dient das `%`-Zeichen als Platzhalter für die restlichen Namensbestandteile. Wir hatten das bereits im Rahmen des Kapitels 4 angesprochen, als es um die Sortierung der Textfelder im Rahmen der `SELECT`-Anweisung ging. Das Ergebnis kann wie in Abbildung 8.3 aussehen.

Abbildung 8.3
Sortierreihenfolgen für den Zeichensatz latin1


Eingabeaufforderung - mysql -uroot
mysql> show collation like 'latin1%';
Collation | Charset | Id | Default | Compiled | Sortlen
latin1_german1_ci | latin1 | 5 | | | 0
latin1_swedish_ci | latin1 | 8 | Yes | Yes | 1
latin1_danish_ci | latin1 | 15 | | | 0
latin1_german2_ci | latin1 | 31 | | Yes | 2
latin1_bin | latin1 | 47 | | Yes | 1
latin1_general_ci | latin1 | 48 | | | 0
latin1_general_cs | latin1 | 49 | | | 0
latin1_spanish_ci | latin1 | 94 | | | 0
8 rows in set (0.00 sec)
mysql>

Grundsätzlich wird zwischen drei Sortierreihenfolgen unterschieden:

- Binäre Sortierung (`_bin`). Es wird nach der Nummer des Zeichens im Zeichensatz sortiert. Hat beispielsweise der Punkt „.“ eine kleinere Nummer als das „A“, wird der Text „.com“ vor „A-Team“ sortiert. Hier ist es wichtig, die genaue Codierung des verwendeten Zeichensatzes zu beachten.
- Alphabetische Sortierung mit Beachtung der Groß-/Kleinschreibung (`_cs`), auch als case-sensitiv bezeichnet.
- Alphabetische Sortierung ohne Beachtung der Groß-/Kleinschreibung, case-insensitiv (`_ci`). Hier gibt es dann länderspezifische Standards. In Deutschland wird dies in der DIN-Norm 5007 festgelegt.

DIN 5007

Die beiden gängigen deutschen Sortierschemata *latin1_german1_ci* und *latin1_german2_ci* entsprechen den in der DIN 5007 festgelegten Sortierungen. Der wesentliche Unterschied zwischen den beiden Sortierungen ist, dass in der ersten Norm, der sogenannten DIN 5007-1, die Umlaute wie die entsprechenden Vokale sortiert werden, also ä wie a, ö wie o und ü wie u. Die DIN 5007-2 betrachtet ä dagegen als ae, ö als oe und ü als ue. Dadurch kann sich eine Sortierreihenfolge im Einzelfall ändern. Während die DIN 5007-2 im Bereich von Namenslisten Verwendung findet, ist sonst weitgehend DIN 5007-1 gebräuchlich.

Eine Angabe könnte also beispielsweise lauten:

```
CREATE SCHEMA IF NOT EXISTS artikel
DEFAULT CHARACTER SET latin1
DEFAULT COLLATE latin1_german1_ci;
```

Betrachten Sie die beiden Begriffe „Schale“ und „schälen“, so beginnt der Unterschied beim Vokal „a“ gegenüber „ä“. Laut DIN 5007-1 (*latin1_german1_ci*) wird „ä“ wie „a“ behandelt. Die Groß- und Kleinschreibung spielt keine Rolle. Damit sind beide Wörter bis „Schale“ bzw. „schäle“ identisch. Erst das folgende „n“ in „schälen“ führt dazu, dass die Sortierung

Beispiel Sortierreihenfolge

1. `Schale`
2. `schälen`

ist. Demgegenüber ist laut DIN 5007-2 (*latin1_german2_ci*) das „ä“ wie „ae“ zu behandeln. Damit unterscheiden sich beide Begriffe nach dem „Scha“ mit dem folgenden „l“ einerseits und „scha“ mit dem folgenden „e“ vom „ae“ andererseits. Da das „e“ vor dem „l“ liegt, werden die Begriffe als

1. `schälen`
2. `Schale`

sortiert.

Nachdem wir Schemata anlegen können, ist es sinnvoll, überflüssige Schemata wieder zu entfernen. Das Gegenstück zu jeder `CREATE`-Anweisung in der SQL-DDL ist immer die `DROP`-Anweisung.

DROP SCHEMA

Entsprechend können Sie ein Datenbankschema komplett mit allen Daten mit der SQL-Anweisung

DROP SCHEMA [IF EXISTS] *schemaname;*

oder gleichwertig

DROP DATABASE [IF EXISTS] *schemaname;*

entfernen. Probieren Sie es doch einmal mit dem oben angelegten Schema `artikel2` aus.

8.1.2 MS Access

MS Access

MS Access hat hinsichtlich des Datenbankschemas eine eingeschränkte Funktionalität. Eine MS Access-Datenbank beinhaltet ein Datenbankschema, das alle im Datenbankfenster sichtbaren Informationen umfasst. Datenbank und Datenbankschema fallen zusammen und werden direkt mit der Anlage einer neuen Datenbank erzeugt. Beim Start von MS Access kann nicht nur eine bestehende Datenbank geöffnet, sondern auch eine neue Datenbank erzeugt werden. Dazu wählen Sie den ersten Punkt LEERE ACCESS-DATENBANK und MS Access erzeugt ein neues Datenbankschema, ohne dass die SQL-Anweisung `CREATE SCHEMA` transparent wird.

Sie können auch eine neue Datenbank anlegen, wenn MS Access bereits geöffnet ist. Mit der Funktion DATEI/NEU öffnet MS Access ein Fenster, in dem Sie entweder eine komplett leere Datenbank neu erzeugen können oder eine der bereits existierenden Musterdatenbanken als Vorlage verwenden können. In jedem Fall wird die aktuelle Datenbank geschlossen, da jeweils immer nur eine Datenbank gleichzeitig bearbeitet werden kann.

Eine Datenbank mit genau einem Datenbankschema besteht in MS Access physikalisch aus einer Datei mit der Endung *.mdb*. Daher öffnet MS Access jetzt ein Fenster, in dem Sie angeben müssen, wo diese Datei gespeichert werden soll und wie sie heißen soll. MS Access schlägt dabei direkt die Endung *.mdb* vor.

Wählen Sie als Name *auftrag* und speichern Sie die neue Datenbank in einem geeigneten Verzeichnis.

Nach dem Bestätigen mit ERSTELLEN haben Sie eine neue, leere Datenbank erzeugt. Sie enthält noch keinerlei Tabellen oder andere Informationen.

8.1.3 Oracle

In Oracle wird mit der Erzeugung eines Users auch ein (leeres) Schema erzeugt. Die entsprechende Anweisung lautet:

```
CREATE USER artikel IDENTIFIED BY pwartikel;
```

Dabei sollten Sie mit dem User *SYS* (oder einem anderen User mit Administrationsrechten) angemeldet sein. Der User und das zugehörige Schema heißen nach der Ausführung der Anweisung `artikel` und das zugehörige Passwort wird hier auf *pwartikel* gesetzt.

Sie müssen dem User jetzt noch das Recht einräumen, in dem Schema neue Tabellen und andere Objekte anzulegen. Am einfachsten erreichen Sie dies mit:

```
GRANT ALL PRIVILEGES TO artikel;
```

Beachten Sie bitte, dass die Beispiele hier für eine lokale Nutzung gedacht sind. In einem größeren Oracle-System werden Sie diesen Befehl nicht absetzen dürfen. Fragen Sie Ihren Administrator nach einem Schema, das Sie beliebig gestalten können.

Jetzt können Sie innerhalb des Schemas arbeiten. Dazu müssen Sie sich abmelden und mit dem eben erzeugten User *artikel* neu anmelden. Jetzt können die Tabellen und andere Objekte angelegt werden, verändert und gelöscht werden. Sie können jetzt auch die Anweisung

```
CREATE SCHEMA AUTHORIZATION artikel;
```

verwenden. Dies mag merkwürdig erscheinen, haben wir doch bereits ein Schema erzeugt und uns in diesem angemeldet. Tatsächlich erzeugt in Oracle diese Anweisung auch kein Schema. Dieses wird bereits von der `CREATE USER`-Anweisung generiert. Die Anweisung `CREATE SCHEMA` dient nur der Bündelung weiterer `CREATE`-Anweisungen, also etwa:

```
CREATE SCHEMA AUTHORIZATION artikel
  CREATE TABLE tabelel1 ...
  CREATE TABLE tabelle2 ...
  CREATE VIEW view1
;
```

Der Vorteil gegenüber der einzelnen Eingabe der CREATE-Anweisungen liegt darin, dass eine CREATE SCHEMA-Anweisung als Transaktion ausgeführt wird, also entweder vollständig oder überhaupt nicht. Tritt also in der Anweisung ein Fehler auf, wird die komplette Anweisung zurückgesetzt, kann korrigiert und erneut ausgeführt werden.

8.1.4 Firebird

Firebird verwendet wie MySQL die Begriffe DATABASE und SCHEMA synonym. Die entsprechende Anweisung lautet von einigen Sonderfunktionen abgesehen:

CREATE {DATABASE|SCHEMA} *dateiname*

[USER *benutzername [PASSWORD passwort]]*

[DEFAULT CHARACTER SET *zeichensatz];*

Sie sehen, dass die Schlüsselwörter DATABASE und SCHEMA wahlweise angegeben werden können. Der dateiname muss ein gültiger Dateiname mit eventuellen Pfadangaben sein, der der physischen Speicherung der Daten dient. Er legt gleichzeitig den Datenbank- und den Schemanamen fest. Ein Benutzer kann aus Sicherheitsgründen verwendet werden. Schließlich kann noch der Standardzeichensatz eingesetzt werden. Wird keiner angegeben, wird der jeweilige Zeichensatz des Betriebssystems verwendet.

Beispiel

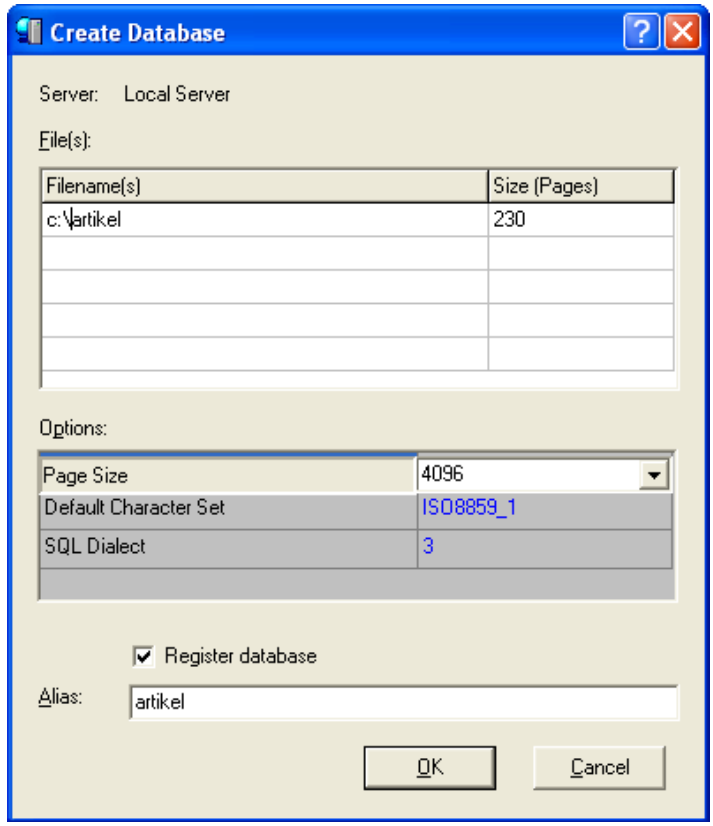
Die Anlage der neuen Artikeldatenbank kann beispielsweise mit folgendem Befehl erfolgen:

```
CREATE SCHEMA 'artikel.fdb' DEFAULT CHARACTER SET ISO8859_1;
```

Ein solcher Befehl kann für Firebird im Hilfswerkzeug isql eingegeben werden, auf das hier aber nicht detailliert eingegangen werden soll. Eine neue Datenbank kann mit Firebird auch über die Oberfläche erzeugt werden. Gehen Sie dazu in der Oberfläche über DATABASE/CREATE DATEBASE. Sie erhalten ein neues Fenster wie in Abbildung 8.4 angegeben. Tragen Sie den Dateinamen mit Pfad ein. Mit der Pagesize geben Sie die Größe der Datei vor (230 ist Minimum). Die Registrierung sollte erfolgen, damit die neue Datenbank beim Firebird-Server bekannt ist. Ein Alias ist der Schemaname, der nach Möglichkeit dem Dateinamen entsprechen sollte (ab dem senkrechten Strich im oberen Fenster, der nicht zum Dateinamen gehört, sondern nur der Markierung in der Abbildung dient).

Damit kann die neue Datenbank als neues Schema erzeugt werden.

Abbildung 8.4
Neue Datenbank/neues Schema in Firebird



8.1.5 openBase

Die Erstellung eines neuen Schemas in openBase entspricht weitgehend der Vorgehensweise in MS Access. Eine Möglichkeit besteht darin, nach dem Start von openBase im Startfenster direkt die Option NEUE DATENBANK ERSTELLEN zu wählen. Der Assistent führt dann die wenigen weiteren Schritte durch.

Die zweite Möglichkeit besteht darin, bei bereits geöffneter Datenbank in openBase mit der Option DATEI/NEU/DATENBANK eine neue Datenbank zu erstellen. Sie werden wiederum vom Assistenten weitergeführt und müssen, wie oben, einen Dateinamen eingeben. Dieser Dateiname ist zugleich eine Datei mit der Endung *.odb*, der physische Speicherort der Datenbank und der Name der Datenbank und des Schemas.

Danach schließt openBase eine andere eventuell geöffnete Datenbank, weil openBase wie MS Access immer nur eine Datenbank mit einem Schema gleichzeitig bearbeiten kann. Anschließend wird die neue noch leere Datenbank geöffnet.

8.1.6 Übungen

1. Ermitteln Sie in einem Wörterbuch, beispielsweise dem Duden, welche Sortierung verwendet wird. Sie können die Begriffe „Schale“ und „schälen“ verwenden. (Ü8.1.1)
2. Ermitteln Sie in Ihrem Telefonbuch die Sortierreihenfolge. Verwenden Sie beispielsweise die Namen „Müller“ und „Mull“. (Ü8.1.2)

8.2 Tabellen erstellen (CREATE TABLE)

8.2.1 Standardangaben für Felder

Nachdem das Datenbankschema erstellt ist, muss es mit Tabellen gefüllt werden, um dann darin die Daten ablegen zu können.

Info

MS Access unterstützt die folgenden Anweisungen nicht. Stattdessen wird eine grafische Oberfläche für die Tabellenerstellung sowie das Mittel der Aktualisierungsabfragen angeboten. openBase unterstützt die Anweisungen und bietet zusätzlich ebenfalls eine grafische Oberfläche. Für die ausschließlichen Nutzer dieser Oberflächen sind die entsprechenden Hinweise am Ende dieses Kapitels zusammengefasst.

Trotzdem lohnt sich auch für diese Benutzer (hoffentlich) das Lesen des Kapitels, da an der einen oder anderen Stelle darauf hingewiesen wird, wie die entsprechenden Anweisungen mit der grafischen Oberfläche zusammenhängen, und so hoffentlich deren Funktionsweise etwas deutlicher wird.

Beispiel

Wir wollen zunächst eine kleine Tabelle für die Umsatzsteuer anlegen, die wir in Kapitel 7 bereits modelliert haben. Zur Erinnerung ist der entsprechende Ausschnitt des Entity-Relationship-Modells in Abbildung 8.5 wiedergegeben.

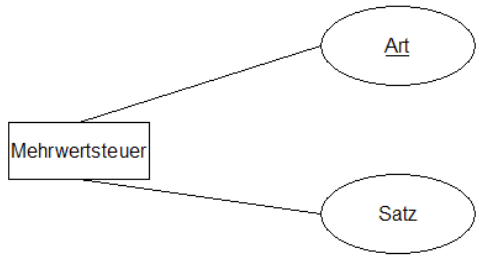


Abbildung 8.5
ERM-Ausschnitt für die Mehrwertsteuer

Daraus ist als Tabellenbeschreibung dann

```
mwst (mwst_art ENUM('voll','ermässigt','kein'), mwst_satz INT)
```

geworden. Um diese Beschreibung in die Datenbank umzusetzen, benötigen wir eine neue SQL-Anweisung: `CREATE TABLE`. Mit der folgenden SQL-Anweisung wird die Tabelle erzeugt:

Listing 8.1
Erzeugen der Tabelle mwst

```
CREATE TABLE mwst(
mwst_art CHARACTER(9) NOT NULL PRIMARY KEY,
mwst_satz SMALLINT DEFAULT 0 NOT NULL);
```

Damit wird eine Tabelle `mwst` angelegt, die zwei Felder beinhaltet. Das erste Feld heißt `mwst_art`. Es kann bis zu 9 Zeichen Text speichern, was für die drei Werte „voll“, „ermässigt“ oder „kein“ ausreicht. Das zweite Datenfeld, `mwst_satz`, wird mit dem Datentyp `SMALLINT` definiert, also als kleine Ganzzahl. Durch die Angabe `NOT NULL` werden `NULL`-Werte verboten, das Feld muss also in jedem Datensatz einen Wert beinhalten, was bei einem Primärschlüssel natürlich sinnvoll ist. `NULL`-Werte können durch die Angabe `NULL` erlaubt werden. Als letzte Angabe hat das erste Attribut den Zusatz `PRIMARY KEY` erhalten, wodurch es zum Primärschlüssel der Tabelle deklariert wird. Der Primärschlüssel kann aus einem oder mehreren Feldern bestehen. Jedes Feld, das Teil eines Primärschlüssels werden soll, muss diesen Zusatz erhalten. Damit haben wir die Modellierung aus dem ERM genau umgesetzt. Die Namen sind abgekürzt und Namensbestandteile durch einen Unterstrich verbunden. Wir haben hinsichtlich der Datentypen – ganzzahlig einerseits und Zeichenkette andererseits – weitere Detailentscheidungen getroffen. Ebenso haben wir entschieden, dass beide Felder keine `NULL`-Werte beinhalten dürfen. Derartige Entscheidungen ergänzen die Modellierung aus dem ERM, sofern sie nicht dort durch entsprechende Erweiterungen bereits festgelegt wurden.

CREATE TABLE

Die Syntax einer `CREATE TABLE`-Anweisung kann also zunächst lauten:

CREATE TABLE *tabellenname (Felddefinition {[,Felddefinition]});*

Die Definition einer Tabelle ist also eine kommaseparierte Liste von Felddefinitionen. Dies entspricht dem Gedanken, dass eine Tabelle zunächst einfach eine sortierte Menge von Feldern ist. Jedes einzelne Feld besitzt eine Felddefinition, die wiederum einen eigenen festen Aufbau hat:

Felddefinition

Felddefinition:

feldname datentyp

[DEFAULT *standardwert]*

[NULL| NOT NULL]

[UNIQUE | PRIMARY KEY]

Feldname

Der Feldname muss dabei den üblichen Regeln für die Vergabe von Namen folgen, also mit einem Buchstaben beginnen, und darf außer einem Unterstrich keine Sonderzeichen beinhalten. Er muss innerhalb einer Tabelle eindeutig sein.

Datentyp

Die Angabe des Datentyps muss einen für das Datenbankmanagementsystem gültigen Datentyp beinhalten. Die Standardtypen für SQL sind in Kapitel 5.1 beschrieben. Gängig sind `CHAR`, `VARCHAR`, `INT`, `SMALLINT`, `FLOAT`, `DOUBLE`, `DATE` oder `TIMESTAMP`. Unter den einzelnen Datenbanksystemen gibt es hier allerdings erhebliche Unterschiede. Oft werden die Datentypen entsprechend dem

zu erwartenden Speicherbedarf noch unterteilt, beispielsweise in TINYINT, SMALLINT, INTEGER und BIGINT. Obige Datentypen sind – bis auf die erwähnten Besonderheiten in MS Access – in den hier verwendeten Datenbanksystemen verfügbar.

Natürlich können Sie auch die Erweiterungen der einzelnen Datenbanksysteme nutzen. So kann alternativ bei Datenbanken wie MySQL, die auch Aufzählungstypen kennen, die Tabelle mit

Listing 8.2
Erzeugen der Tabelle mwst mit einem Aufzählungstyp für das Feld mwst_art

```
CREATE TABLE IF NOT EXISTS mwst(
mwst_art ENUM('voll','ermässigt','kein') NOT NULL PRIMARY KEY,
mwst_satz SMALLINT NOT NULL DEFAULT 0
);
```

definiert werden. Damit wird eine Tabelle mwst angelegt, die wiederum zwei Felder beinhaltet. Das erste Feld, mwst_art, kann zwar wieder Texte aufnehmen, diese können aber nur die drei Werte „voll“, „ermässigt“ oder „kein“ sein, was an dem Aufzählungstyp ENUM erkennbar ist.

NOT EXISTS

Mit dem Zusatz IF NOT EXISTSdirekt nach dem CREATE TABLEkann verhindert werden, dass das Datenbanksystem einen Fehler erzeugt, wenn bereits eine Tabelle mit demselben Namen vorhanden ist. Gerade bei der automatisierten Erzeugung kompletter Schemata kann das von Vorteil sein, wenn es vom Datenbanksystem unterstützt wird.

NULL

Die Angabe NOT NULL erzwingt eine Werteingabe für ein Feld. Interessant ist die Kombination mit der DEFAULT-Option. Wird ein DEFAULT-Wert angegeben, wird dieser Wert bei fehlender Eingabe als Standardwert verwendet. Damit erfolgt automatisch eine Eingabe und die NOT NULL-Bedingung ist in jedem Fall erfüllt.

DEFAULT

Für die DEFAULT-Angabe sind laut Standard erlaubt:

- ein beliebiges Literal, also ein Wert wie „0“, „nicht vorhanden“ oder „unbekannt“
- eine Funktion zur Datumsermittlung
- USER
- CURRENT_USER
- SESSION_USER
- SYSTEM_USER
- NULL

Die Datums- und USER-Angaben erlauben einen dynamischen Eintrag gängiger Werte, also hier je nach Datenbank verschiedene Formate des aktuellen Datums, der Uhrzeit sowie des angemeldeten Benutzers, ohne dass sich der Anwender oder ein Programm darum kümmern muss. Die Angabe von NULL stellt eine Besonderheit dar. Beachten Sie mögliche Konflikte mit der NOT NULL-Bedingung.

Die meisten Implementierungen lassen aber bei Weitem nicht alle Optionen zu. So können Sie bei MySQL außer Literalen nur für TIMESTAMP-Felder die Funktion CURRENT_TIMESTAMP nutzen, um das aktuelle Datum für alle Datensätze einzutragen.

Beachten Sie auch, dass die Reihenfolge der Angabe von NULL oder NOT NULL einerseits und DEFAULT andererseits bei einigen Datenbanksystemen eine Rolle spielt.

PRIMARY KEY

Mit der Option PRIMARY KEY wird ein Feld zum Primärschlüsselfeld der Tabelle. Damit muss jeder Wert dieses Feldes in der gesamten Tabelle eindeutig sein, kein Datensatz darf denselben Wert wie ein anderer Datensatz haben.

UNIQUE

Im vorherigen Kapitel haben wir im Zusammenhang mit den Alternativschlüsseln gesehen, dass es neben dem eigentlichen Primärschlüssel auch weitere Schlüsselkandidaten geben kann. Sind diese nicht im Rahmen der Normalisierung entfernt worden, können die entsprechenden Felder mit der Option UNIQUE ebenfalls als eindeutig definiert werden. Das Datenbankmanagementsystem wird diese Eindeutigkeit dann bei jedem neuen (INSERT) oder geänderten (UPDATE) Datensatz überprüfen.

UNIQUE

Ein Primärschlüssel erfordert eindeutige Werte in allen Datensätzen einer Tabelle. Damit kann durch Angabe der Werte des oder der Primärschlüsselfelder jeder Datensatz einer Tabelle eindeutig identifiziert werden.

Neben dem Primärschlüssel können auch die Angaben in weiteren Feldern über eine gesamte Tabelle eindeutig sein. Es handelt sich um die Schlüsselkandidaten. Soll diese Eindeutigkeit ebenfalls sichergestellt werden, kann eine UNIQUE-Bedingung mit derselben Syntax wie der PRIMARY KEY mit dem Schlüsselwort UNIQUE definiert werden.

Beispiel

Wir wollen als weiteres Beispiel die Kundentabelle anlegen. Dazu nutzen wir folgende SQL-Anweisung:

Listing 8.3 Erstellen der Kundentabelle

```
CREATE TABLE kunden (
   mid INT NOT NULL,
   kid VARCHAR(50) NOT NULL,
   firma VARCHAR(59) UNIQUE,
   anrede VARCHAR(10) NULL,
   nachname VARCHAR(50) NOT NULL,
   vorname VARCHAR(30) NULL,
   rechnungsadresse VARCHAR(255) DEFAULT 'wie Anschrift' NOT NULL ,
   strasse VARCHAR(50) NULL,
   PLZ CHAR(5) NULL,
   ort VARCHAR(50) DEFAULT 'Celle' NOT NULL ,
   land VARCHAR(50) DEFAULT 'Deutschland' NULL,
   geworbenvon INT NULL,
   geburtsdatum DATE NULL,
   letzte_aenderung TIMESTAMP DEFAULT current_timestamp,
   CONSTRAINT PKKD PRIMARY KEY (mid,kid)
);
```

Primärschlüssel

Dies entspricht der Entität *Kunde*, wie wir sie in Kapitel 7 modelliert haben. Sie sehen, dass nicht nur die Kunden-ID (kid) als Primärschlüsselfeld markiert ist, sondern auch die Mandanten-ID (mid), das wird in Abschnitt 8.3.1 näher erläutert. Dies ist die unmittelbare Übernahme der Überlegung aus der

Modellierung, dass Kunden über die verschiedenen Firmen der BüroFix (Mandanten) hinweg eindeutig sein sollen. Anderenfalls hätte das Feld `mid` nicht in den Primärschlüssel aufgenommen werden dürfen. Die `kid` ist im Gegensatz zur `mid` nicht als `INT`, also als Ganzzahl, sondern als `VARCHAR(50)` definiert. Das bedeutet, dass alphanumerische Bezeichnungen wie „Müller, Hannover-Bothfeld“ erlaubt sind. Maximal können hier Einträge aus 50 Zeichen gespeichert werden. Um Speicherplatz zu sparen, ist das Feld als `VARCHAR`, also als variables Zeichenfeld (Character), definiert. Dies ist eine eher ungewöhnliche Modellierung, da sie im Allgemeinen als Primärschlüssel nicht gerade performant, aber grundsätzlich möglich ist.

Weitere Felder

Die `firma` ist ebenfalls als `VARCHAR` gespeichert. Da hier nicht dieselbe Firma mehrmals erscheinen soll, ist das Feld mit `UNIQUE` als eindeutig definiert worden. Das Feld `Rechnungsadresse` hatten wir bereits im Rahmen der Modellierung problematisiert, aber es war gewünscht. Hier wird jetzt ein längeres Feld mit bis zu 255 Zeichen vorgesehen, um auch längere Adressen komplett speichern zu können. Da in vielen Fällen keine eigene Rechnungsadresse zu erwarten ist, soll hier als Standardwert der Eintrag „wie Anschrift“ eingesetzt werden. Beim Einfügen eines neuen Kunden reicht dann die Angabe `DEFAULT`, um diesen Wert zu erhalten. Entsprechend werden Ort und Land mit „Celle“ respektive „Deutschland“ vorbelegt.

Timestamp

Eine Besonderheit stellt das Feld `letzte_aenderung` dar. Es war in der logischen Modellierung des ERM nicht vorgesehen. Andererseits ist es gerade bei Stammdaten wichtig, zu erkennen, wann diese zuletzt aktualisiert wurden. Hier wird daher ein Feld aufgenommen, in dem ein `TIMESTAMP`, also ein Zeitstempel, abgelegt wird, aus dem Datum und Uhrzeit der letzten Änderung ersichtlich sind. Um sicherzustellen, dass das Feld beim Anlegen eines Datensatzes aktualisiert wird, kann eine Funktion `CURRENT_TIMESTAMP` (Syntax in MySQL, Oracle, Firebird und openBase) genutzt werden, um den Wert in jedem Fall einzutragen.

Firebird

Bei Firebird ist generell zu beachten, dass der „Feuervogel“ Wert auf seine Eigenintelligenz legt. Wenn Sie also ein Datenfeld ohne `NULL`-Werte erzeugen wollen, müssen Sie dies mit `NOT NULL` angeben. Anderenfalls denkt sich Firebird, dass Sie `NULL`-Werte zulassen, und möchte nicht, dass Sie ihm dies durch ein gesondertes `NULL` noch einmal bestätigen. Also, wenn Sie `NULL`-Werte zulassen wollen, einfach nichts angeben.

openBase

Bei openBase sollte kein `UNIQUE` angegeben werden. Außerdem ist bei openBase das Thema der Groß-/Kleinschreibung zu beachten. Wird der Name der Tabelle oder eines Datenfeldes nicht in Anführungszeichen gesetzt, wird er in Großbuchstaben umgewandelt. Insofern verhält sich openBase standardkonform. Andererseits erlaubt es die Verwendung von Anführungszeichen, um eine Groß-/Kleinschreibung zu erzwingen. Hier müssen Sie entscheiden, was Ihnen lieber ist. Die Beispiele der Artikeldatenbank sind ohne Anführungszeichen angegeben. Wenn Sie die Groß-/Kleinschreibung verwenden, müssen Sie bei allen anderen Angaben in dem Fenster SQL STATEMENT AUSFÜHREN die Anführungszeichen ebenfalls verwenden.

Tabellenerstellungsabfrage in MS Access

MS Access kennt das Mittel der Tabellenerstellungsabfrage. Dabei kann eine normale Abfrage erstellt werden, die dann in der Entwurfsansicht über den Menüpunkt ABFRAGE/TABELLENERSTELLUNGSABFRAGE in eine Tabellenerstellungsabfrage umgesetzt wird. MS Access fragt ab, wie die neue Tabelle heißen soll, die die Abfrage erstellt. Dabei kann noch entschieden werden, ob nur die Tabellenstruktur erstellt wird (entspricht `CREATE TABLE`) oder ob auch die Daten mit übernommen werden sollen (`INSERT`).

Im Kern ermittelt MS Access aus den Angaben der ausgewählten Felder deren Eigenschaften, wie Datentyp, `NOT NULL` und weitere Angaben, und erzeugt gleichnamige Felder mit identischen Eigenschaften in der neuen Tabelle. Da alle Informationen vorhanden sind, kann dies vollautomatisch geschehen. Die entsprechende `CREATE TABLE`-Anweisung wird nicht sichtbar. Durch Angabe eines Ausdrucks können die neuen Felder auch umbenannt werden, also etwa *Neuer_Name:[Alter_Name]* in der Zeile FELD.

Zusätzlich können die Daten aus der Abfrage auch sofort in die neue Tabelle übernommen werden. MS Access generiert daraus:

SELECT *feldname, feldname INTO neue_tabelle FROM alte_tabelle WHERE ...;*

Der Vorteil dieses Vorgehens ist, dass gleichartige Felder nicht erneut beschrieben werden müssen. Andererseits besteht hier immer die Gefahr, redundante Felder zu erzeugen. Felder mit wirklich neuem Inhalt müssen zumeist doch manuell angelegt werden.

Wirkliche Tabellenerstellungsabfragen lassen sich leider nur über die Programmierschnittstelle der Jet Engine realisieren.

Übungen

Übungen zur einfachen CREATE TABLE-Anweisung

1. Erstellen Sie die Tabelle `warengruppe` gemäß der Definition aus Kapitel 7 im Schema `artikel`. Ist der Name der Warengruppe nicht bekannt, soll er „unbekannt“ sein. Er soll aber in jedem Fall angegeben sein. (Ü8.2.1.1)
2. Erstellen Sie ein neues Schema (eine neue Datenbank) mit dem Namen `Kap8`. (Ü8.2.1.2)
3. Erstellen Sie in dem neuen Schema eine Tabelle `fahrzeug`, in der das Kfz-Kennzeichen, die Fahrgestellnummer, die Automarke, die Farbe des Fahrzeugs (Standard ist Silber), die Anzahl der Türen, der Hubraum, der jährliche Versicherungsbeitrag sowie Datum und Uhrzeit der letzten Änderung angegeben werden. Das Kennzeichen, die Fahrgestellnummer, die Automarke und der Versicherungsbeitrag sind Pflichtangaben. Das Kennzeichen ist der Primärschlüssel. (Ü8.2.1.3)
4. Erstellen Sie eine Tabelle `lieferung` für die Angaben in Abbildung 8.6. (Ü8.2.1.4)
5. Welche Probleme sehen Sie in der Tabelle in Abbildung 8.6? (Ü8.2.1.5)

RNr	Name	Vorname	PLZ	Ort	Strasse	Telefon	KNr	Preis	Datum	Anz	Lieferung	Fahrer
1	Timme	Ulf	29223	Celle	Rosenstraße 117	342565	1	14,00 €	03.10.2006	2	Spinaci, Salame	Michael
2	Petersen	Annemarie	29221	Celle	Neustadt 3	897654	2	40,00 €	03.10.2006	3	Salame, Quattro stagione, Haw	Christian
3	Havenstein	Katrin	29225	Celle	Welfenallee 43	897643	3	7,00 €	04.10.2006	1	Salame	Michael
4	Timme	Ulf	29223	Celle	Rosenstraße 117	342565	1	9,00 €	04.10.2006	2	Spinaci, Regina	Michael
5	Timme	Ulf	29223	Celle	Rosenstraße 117	342565	1	12,00 €	02.10.2006	2	Hawaii, Spinaci	Christian

Abbildung 8.6
Tabelle zu den Übungsaufgaben

8.2.2 Fremdschlüsselbeziehungen

Die `CREATE TABLE`-Anweisung beinhaltet in den Felddefinitionen der einzelnen Felder die zum Aufbau der Tabelle notwendigen Informationen. Eine besondere Rolle spielen dabei die Fremdschlüsselattribute, die die Beziehungen des Entity-Relationship-Modells in Beziehungen zwischen den Tabellen umsetzen. Das folgende Beispiel zeigt die Erstellung der Tabelle `artikel` mit dem gleichzeitigen Aufbau von Beziehungen zu den Tabellen `warengruppe` und `mwst` (Mehrwertsteuersatz). Schauen wir uns dazu zunächst die Situation der Entität *Artikel* mit ihren Attributen und Beziehungen im ERM an (Abbildung 8.7).

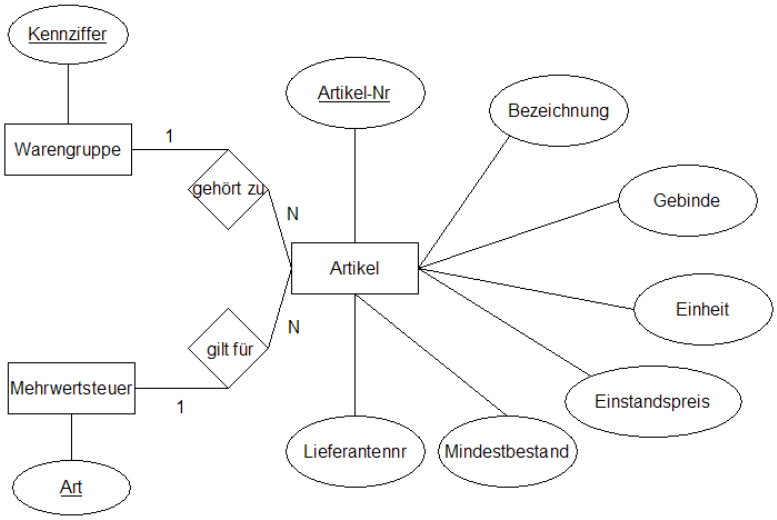


Abbildung 8.7
Die Entität Artikel mit ihren Attributen und Beziehungen

Die Umsetzung in eine SQL `CREATE`-Anweisung ist jetzt etwas komplexer als bisher. Es sind nicht nur die Attribute zu berücksichtigen, sondern auch die Beziehungen zu den anderen Entitäten *Warengruppe* und *Mehrwertsteuer*. Beim `CREATE TABLE` sind diese Beziehungen umzusetzen, um die Daten aus den Tabellen für die Artikel, Mehrwertsteuer und Warengruppe zusammenhängend bearbeiten zu können. Die SQL-Anweisung dazu ist in Listing 8.4 dargestellt.

```
CREATE TABLE artikel (
 anr int NOT NULL PRIMARY KEY,
 bezeichnung VARCHAR(255) NOT NULL,
 gebinde FLOAT,
 einheit VARCHAR(30),
 wgknz INT DEFAULT 0 NOT NULL REFERENCES warengruppe(kennziffer),
```

Listing 8.4
Erstellen der Tabelle artikel mit zwei Fremdschlüsselbeziehungen

Listing 8.4 (Forts.) Erstellen der Tabelle artikel mit zwei Fremdschlüsselbeziehungen

```
einstandspreis DECIMAL(6,2),
listenpreis DECIMAL(6,2),
mwst_art CHAR(9) DEFAULT 'voll' NOT NULL REFERENCES mwst(mwst_art),
bestand INT DEFAULT 0 NOT NULL,
letzte_aenderung timestamp DEFAULT current_timestamp);
```

Beachten Sie insbesondere die Zeilen

```
wgknz INT DEFAULT 0 NOT NULL REFERENCES warengruppe(kennziffer)
```

und

```
mwst_art CHAR(9) DEFAULT 'voll' NOT NULL REFERENCES mwst(mwst_art)
```

in denen die Fremdschlüsselbeziehungen aufgebaut werden, die die beiden Beziehungen in Abbildung 8.7 umsetzen.

In der ersten `REFERENCES`-Zeile wird das Feld `wgknz` erzeugt, also die Warengruppenkennziffer. Dabei wird mit dem Schlüsselwort `REFERENCES` gleichzeitig eine Fremdschlüsselbeziehung zum Feld `kennziffer` der Tabelle `warengruppe` aufgebaut. Nach dem Schlüsselwort `REFERENCES` wird dabei zunächst der Tabellenname der zweiten Tabelle angegeben, danach in Klammern das Feld, das mit `wgknz` in Beziehung gesetzt werden soll. Die Zeile bewirkt damit, dass die Datenbank weiß, dass Werte dieser beiden Felder übereinstimmen sollen und wahrscheinlich öfter in `SELECT`-Abfragen verwendet werden. Die Beziehung zwischen den Entitäten *Artikel* und *Warengruppe* ist implementiert.

In der zweiten dieser beiden Zeilen wird das Feld `mwst_art` erzeugt. Mit dem Zusatz `REFERENCES` wird wieder eine Fremdschlüsselbeziehung aufgebaut. Das Feld `mwst_art` wird in Beziehung zu dem entsprechenden Feld in der Tabelle `mwst` hergestellt. Die Beziehung zwischen den Entitäten *Artikel* und *Mehrwertsteuer* ist damit ebenfalls implementiert.

Die bisherige Syntax der Felddefinition erweitert sich mit der Möglichkeit, Fremdschlüsselbeziehungen zu anderen Tabellen aufzubauen, somit zu:

Felddefinition:

feldname datentyp

[DEFAULT *standardwert]*

[NULL| NOT NULL]

[UNIQUE | PRIMARY KEY]

[REFERENCES *tabellenname (feldname)]*

Umsetzung der Beziehung

Wir haben zuvor die Beziehungen zwischen den Entitäten *Artikel* und *Warengruppe* sowie *Artikel* und *Mehrwertsteuer* erzeugt. Die Kennziffer ist als Primärschlüssel der Entität *Warengruppe* modelliert. Sie ist bezüglich aller Datensätze der Tabelle eindeutig. Dieses Feld wird als Fremdschlüssel in die Tabelle `artikel` übernommen, die über eine Beziehung verbunden werden. Über die Gleichsetzung

Primärschlüssel einer Tabelle = Fremdschlüssel der anderen Tabelle

oder konkret

warengruppe.kennziffer = artikel.wgknz

kann daher durch die Aufnahme des Fremdschlüsselfeldes `wkgnz` mit den passenden Werten in die Tabelle `artikel` die Beziehung über SQL in die Datenbank umgesetzt werden.

Optionale Beziehung

Die Angabe derartiger Fremdschlüsselbeziehungen mit `REFERENCES` ist nicht zwingend erforderlich, die Beziehungen können auch nur durch die entsprechenden Angaben in den `SELECT`-Anweisungen genutzt werden. Durch Beachtung sinnvoller und zusammenpassender Werte bei `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen können die Beziehungen auch inhaltlich sichergestellt werden. Bei einem manuellen Vorgehen, also dem Verzicht auf die `REFERENCES`-Angaben, liegt die Verantwortung für die Pflege der Beziehungen bei dem Anwender oder dem Programm, das die Daten ändert, einfügt oder löscht. Dies kann gewünscht sein. Es erhöht den Freiheitsgrad bei der Programmierung der Datenbankanwendung, da die Datenbank weniger prüft und somit auch weniger Fehlermeldungen durch fehlende oder nicht zusammenpassende Daten entstehen. Umgekehrt erhöht dies das Fehlerrisiko, da die Datenbank die Qualität der Daten in geringerem Maß selbst sicherstellen kann.

Vorteile des REFERENCES

Andererseits hat die Verwendung der `REFERENCES`-Angabe natürlich auch erhebliche Vorteile. Die Verwendung der `REFERENCES`-Klausel erlaubt der Datenbank die Prüfung auf zusammenhängende Daten, in der obigen Anweisung beispielsweise die Angabe einer gültigen Warengruppe bei der Erfassung eines Artikels. Letztlich können alle im Zusammenhang mit der referenziellen Integrität möglichen Restriktionen durch die Datenbank selbst überprüft werden. Damit kann die Datenbank die Integrität der Beziehungen garantieren und somit zur Datenqualität erheblich beitragen. Ein weiterer Vorteil der Verwendung der `REFERENCES`-Klausel liegt darin, dass die Datenbank, wenn sie um den Zusammenhang von Primärschlüssel- und Fremdschlüsselfeld weiß, eigenständig den Zugriff auf diese Beziehung optimieren kann. So kann beispielsweise ein Index auf den Fremdschlüssel generiert werden, der bei Nutzung der Beziehung einen deutlich schnelleren Zugriff zwischen den beiden Tabellen garantiert.

Übungen

Übungen zur CREATE TABLE-Anweisung mit REFERENCES

1. Erstellen Sie die Tabelle `fahrzeughalter`, die den Namen und Vornamen des Fahrzeughalters sowie einen Fremdschlüssel `kennzeichen` enthält, der sich auf die in Ü8.2.1.3 erstellte Tabelle `fahrzeug` bezieht. Der Primärschlüssel ist der Name. (Ü8.2.2.1)
2. Erstellen Sie eine Tabelle `inspektion`, die eine Nummer und das Datum einer Inspektion zeigt und als Fremdschlüssel das Fahrzeug beinhaltet. (Ü8.2.2.2)

8.3 Integritätsbedingung

Neben der Definition von Feldern können bei der Erzeugung der Tabellenstruktur mit CREATE TABLE weitere Bedingungen angegeben werden. Dabei handelt es sich um Integritätsbedingungen, die das Datenbankmanagementsystem selbstständig überprüft beziehungsweise die es ihm erlauben, den Zugriff auf die Daten zu optimieren. Die wesentlichen Typen sind:

- **Primärschlüssel:** eine alternative Definitionsmöglichkeit des Primärschlüssels, die insbesondere bei mehreren Primärschlüsselattributen sinnvoll ist.
- **Fremdschlüssel:** eine alternative Definitionsmöglichkeit für einen Fremdschlüssel, die insbesondere bei der Nutzung der zusätzlichen Optionen übersichtlicher ist.
- **CHECK:** weitere Bedingungen, die für die Datensätze der Tabelle einzuhalten sind.
- **Indizes:** Optimierung des Zugriffs auf bestimmte Felder durch die bewusste Einrichtung schneller Suchmechanismen für einzelne Felder oder Gruppen von Feldern.

Diese Integritätsbedingungen stehen neben den Felddefinitionen. Sie können wie Felddefinitionen als Aufzählung in die Definition des CREATE TABLE aufgenommen werden. Damit erweitert sich die Syntax für die CREATE TABLE-Anweisung.

Syntax

```
CREATE [IF NOT EXISTS] TABLE (
[Felddefinition|Integritätsbedingung] { ,[Felddefinition|Integritätsbedingung]};
```

Sie sehen, dass Felddefinitionen und Integritätsbedingungen jeweils durch ein Komma getrennt aneinandergereiht werden. Wir wollen die verschiedenen Typen einzeln betrachten.

8.3.1 Primärschlüssel (PRIMARY KEY)

Der Primärschlüssel dient der Identifizierung der Datensätze in einer Tabelle. Jede Tabelle sollte daher einen Primärschlüssel besitzen, auch wenn dies technisch nicht zwingend vorgeschrieben ist. Ein Primärschlüssel besteht aus einem oder mehreren Feldern. Eine Möglichkeit, den Primärschlüssel zu definieren, besteht in dem Zusatz PRIMARY KEY bei der Felddefinition des Primärschlüsselfeldes. Dies ist die einfachste Möglichkeit, einen Primärschlüssel zu definieren, die aber nur möglich ist, wenn der Primärschlüssel aus einem einzigen Feld besteht und keine weiteren Angaben zu machen sind.

Eine zweite Möglichkeit, einen Primärschlüssel zu definieren, ist, eine Integritätsbedingung bei der Tabellendefinition mit CREATE TABLE anzugeben. Diese Variante erlaubt insbesondere die Angabe mehrerer Felder als Primärschlüssel. Sie kann außerdem im Rahmen der ALTER TABLE -Anweisung genutzt werden.

Integritätsbedingung:

Primärschlüssel

[CONSTRAINT *name]* *PRIMARY KEY (feldname [{ , feldname}]*

Damit kann ein einzelnes Feld oder eine Liste von Feldern angegeben werden, die gemeinsam den Primärschlüssel der Tabelle bilden. Wir wollen dies am Beispiel der Entität *Bestellposition* betrachten.

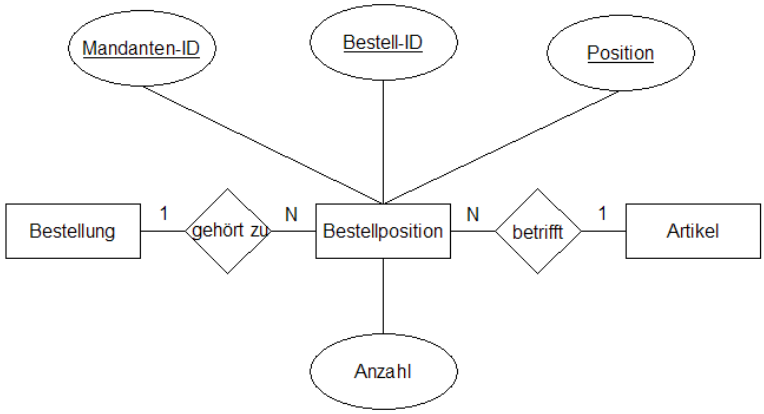


Abbildung 8.8
Umgebung der Entität Bestellposition

Die Entität *Bestellposition* besitzt laut Modellierung drei Primärschlüsselfelder: die Mandanten-ID, die Bestell-ID und die Position. Daher bietet es sich hier an, den Primärschlüssel über eine Integritätsbedingung mit den drei Feldern `mid`, `bnr` und `pos` zu definieren:

Listing 8.5
Erstellen der Tabelle für die Bestellpositionen

```
CREATE TABLE bestell_position (
   mid INT NOT NULL,
   bnr INT NOT NULL,
   pos INT NOT NULL,
   anr INT NOT NULL,
   anzahl INT DEFAULT 1 NOT NULL,
   letzte_aenderung TIMESTAMP DEFAULT current_timestamp,
   CONSTRAINT PKBESTPOS PRIMARY KEY (mid,bnr,pos)
);
```

Mit der letzten Zeile wird ein Primärschlüssel definiert, der aus den drei Feldern `mid`, `bnr` und `pos` besteht, was der Mandanten-ID, der Bestellnummer und der Bestellposition entspricht. Die Integritätsbedingung hat über den Zusatz `CONSTRAINT PKBESTPOS` einen eigenen Namen bekommen. Dieser Zusatz ist nicht zwingend, aber praktisch, wenn der `CONSTRAINT` später anzupassen ist.

Das Ergebnis der obigen Anweisung können Sie mit

```
DESCRIBE bestell_position;
```

überprüfen. In MySQL erhalten Sie dann die Darstellung in Abbildung 8.9.

Abbildung 8.9
Ergebnis der Tabellendefinition der Tabelle bestell_position

Field	Type	Null	Key	Default
mid	int(11)	NO	PRI	
bnr	int(11)	NO	PRI	
pos	int(11)	NO	PRI	
anr	int(11)	NO		
anzahl	int(11)	NO		1
letzte_aenderung	timestamp	YES		CURRENT_TIMESTAMP

Den Sinn der drei Primärschlüsselfelder können Sie in der Abbildung 8.10 erkennen. In einer Bestellung sind die Mandantennummer und die Bestellnummer normalerweise identisch. Daher unterscheiden sich die einzelnen Bestellpositionen nur in der Positionsnummer pos.

Abbildung 8.10
Zwei Datensätze, die sich nur in einem Primärschlüsselfeld unterscheiden

kid	bnr	pos	anr	anzahl	letzte_aenderung
2	4711	1	1002	10	2007-09-23 17:29:24
2	4711	2	9801	5	2007-09-23 17:29:24

8.3.2 Fremdschlüssel erstellen (FOREIGN KEY)

Gleich Feldanzahl in Primär- und Fremdschlüssel

Fremdschlüssel dienen der Verbindung von Tabellen untereinander. Eine Beziehung zwischen zwei Tabellen wird durch eine Festlegung getroffen, die angibt, welches Fremdschlüsselfeld welchem Primärschlüsselfeld in einer anderen Tabelle entspricht. Ein Primärschlüssel kann – wie gesehen – aus einem oder mehreren Feldern bestehen. Entsprechend gilt dasselbe auch für den Fremdschlüssel. Die Anzahl der Felder in Primärschlüssel und Fremdschlüssel muss sich natürlich entsprechen, damit die Felder paarweise verbunden werden können.

Der einfachste Fall eines Primärschlüssels ist ein einziges Feld. Entsprechend wird dann auch nur ein Fremdfeld benötigt. Das ist einfach, daher ist ein solches Konzept oft wünschenswert. In der Praxis zeigen sich aber immer Fälle, wo auch Konzepte mit mehreren Feldern sinnvoll erscheinen.

Tipp

Datenbankadministratoren lieben einfache Primär- und Fremdschlüssel. Sie werden viele Gründe vortragen, warum Sie keine mehrfachen Schlüssel brauchen, und viele dieser Gründe sind stichhaltig. Tatsächlich lässt sich jede logisch modellierte Kombination mehrerer Felder zu einem Schlüssel im Rahmen der Umsetzung in das Datenbankmodell durch ein Feld ersetzen. Es ist in jedem Fall bedenkenswert, ein neues, rein technisches Schlüsselfeld einzuführen, das im einfachsten Fall einfach aus einer Nummerierung besteht.

Dieses Feld kann dann als Primärschlüssel genutzt und ein entsprechendes Feld als Fremdschlüssel in andere Tabellen aufgenommen werden. Es ist performant und einfach.

Eine solche Lösung sollte aber nach Möglichkeit vor dem Anwender der Datenbank verborgen werden, da er mit diesem rein technischen Schlüssel nichts anfangen kann. Können Sie über die Anwendung sicherstellen, dass dies möglich ist, sollten Sie statt des logischen Schlüssels mit mehreren Feldern ein neues Feld als technischen Primärschlüssel verwenden. Die ursprünglichen Schlüsselfelder bleiben natürlich erhalten und können als Alternativschlüssel dienen.

Ist dies nicht möglich, sollte im Interesse des Anwenders doch die ursprüngliche Lösung mit mehreren Schlüsselfeldern bevorzugt werden.

Es können also Situationen auftreten, in denen Schlüssel (und Fremdschlüssel) mit mehreren Feldern sinnvoll sind. Dem wird Rechnung getragen, indem in unserem Beispiel für die Bestellungen mit mehreren Primärschlüsselfeldern gearbeitet wird. SQL unterstützt dieses Konzept im Rahmen der Integritätsbedingung für Fremdschlüssel, indem Feldlisten für Primärschlüsselfelder und Fremdschlüsselfelder zugelassen sind.

Integritätsbedingung: **FOREIGN KEY**

[CONSTRAINT *name]* FOREIGN KEY *(feldname [{ , feldname}]*

REFERENCES *tabellenname (feldname [{ , feldname}]*

[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT| SET NULL}]

[ON DELETE {NO ACTION|CASCADE|SET DEFAULT| SET NULL}]

Nach dem Schlüsselwort `FOREIGN KEY` wird der Fremdschlüssel als Liste der Fremdschlüsselfelder angegeben. Die Fremdschlüsselfelder müssen Felder der Tabelle sein, in der die Integritätsbedingung angegeben wird. Über das nachfolgende Schlüsselwort `REFERENCES` wird der Bezug zu der anderen Tabelle angegeben, in der anschließend der Primärschlüssel als Liste der Primärschlüsselfelder angegeben wird.

Betrachten wir dazu die Tabelle der Bestellungen. **Beispiel**

Listing 8.6
Erstellung einer Tabelle mit referenzieller Integrität

```
CREATE TABLE bestellung (
   mid INT NOT NULL,
   bnr INT NOT NULL,
   kid VARCHAR(50) NOT NULL,
   bestelldatum DATE NOT NULL,
   letzte_aenderung TIMESTAMP DEFAULT current_timestamp,
   CONSTRAINT PKBEST PRIMARY KEY (mid,bnr,kid),
   CONSTRAINT FKBESTKUND FOREIGN KEY (mid,kid) REFERENCES
kunden(mid,kid)
   ON UPDATE CASCADE
   ON DELETE SET NULL
);
```

Über diese Anweisung wird eine Beziehung zwischen den beiden Tabellen `bestellung` und `kunden` hergestellt. Die Tabelle `kunden` enthält den Primärschlüssel, sie wird auch als Primärschlüsseltabelle oder Mastertabelle bezeichnet. **Mastertabelle**

Detailtabelle

Die Tabelle `bestellung` enthält den Fremdschlüssel. Sie wird als Fremdschlüsseltabelle, als Detailtabelle oder als Slave-Tabelle bezeichnet.

Fremdschlüsselbeziehung

Die Beziehung wird hergestellt, indem die Felder in der Reihenfolge ihrer Nennung – nicht ihrer Namen – miteinander in Beziehung gesetzt werden. Im obigen Beispiel entspricht das Fremdschlüsselfeld `mid` dem Primärschlüsselfeld `mid`, weil beide als Erstes in der jeweiligen Liste genannt werden. Entsprechend müssen die Werte der Bestellnummer `kid` in der Tabelle `bestellung` den Werten des Feldes `kid` in der Tabelle `bestell` entsprechen, da beide an zweiter Stelle genannt werden.

Referenzielle Integrität in anderen Datenbanken

Referenzielle Integrität wird in Form der `ON DELETE`- und `ON CASCADE`-Angaben von den einzelnen Datenbanken in unterschiedlicher Form unterstützt. Manche Systeme kennen nur `ON DELETE`, manche gar keine referenzielle Integrität.

In MS Access ist die referenzielle Integrität in die grafische Oberfläche integriert. Hier kann in der Beziehungsansicht jede einzelne Beziehung aktiviert und deren Eigenschaften bearbeitet werden. Die Aktualisierungsweitergabe und die Löschweitergabe entsprechen jeweils dem `ON UPDATE CASCADE` respektive `ON DELETE CASCADE`.

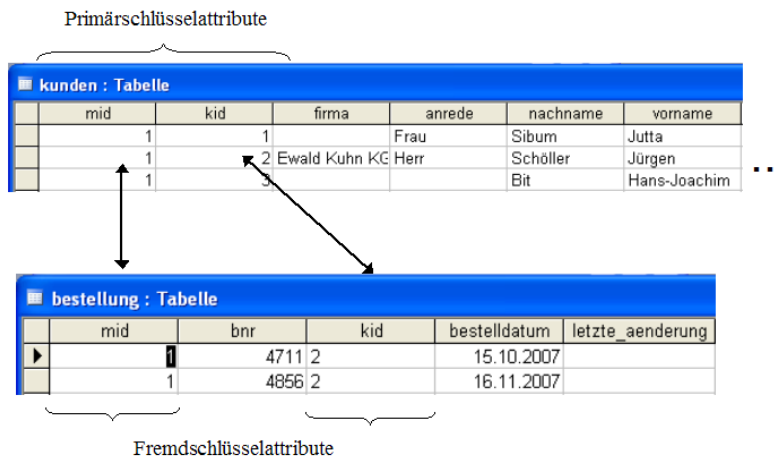


Abbildung 8.11 Beziehung zwischen Tabellen bestellung und kunde

Weil der Primärschlüssel jeden Datensatz einer Tabelle eindeutig identifiziert, kann für jeden Wert des Primärschlüssels stets nur ein Datensatz in der Primärschlüsseltabelle existieren. Demgegenüber kann dieser Wert in mehreren Datensätzen der Fremdschlüsseltabelle auftreten, da deren Datensätze nicht über den Fremdschlüssel identifiziert werden. Es handelt sich also im Normalfall um eine 1:n-Beziehung: Einem Datensatz in der Primärschlüsseltabelle können mehrere Datensätze in der Fremdschlüsseltabelle entsprechen. Wohlgemerkt „können“. Es kann zu einem Datensatz in der Primärschlüsseltabelle auch kein oder nur ein Datensatz in der Fremd-

schlüsseltabelle vorhanden sein. In Abbildung 8.11 sehen Sie zwei Bestellungen der „Ewald Kuhn KG“.

Anomalien

Beziehungen können bei der Bearbeitung von Datenbanken zu den bekannten Problemen von Anomalien führen. So würde das Löschen des Datensatzes für die Ewald Kuhn KG in der Tabelle `kunden` in Abbildung 8.11 zu Problemen mit den Datensätzen in der Tabelle `bestellung` führen. Bestellungen ohne einen zugehörigen Kunden sind nicht sinnvoll und müssten eigentlich entfernt werden. Andererseits kann ihre vollständige Löschung aber auch zu einem unerwünschten Informationsverlust führen. Die Entscheidung, wie in einem solchen Fall vorzugehen ist, kann nicht das Datenbankmanagementsystem treffen. Diese Entscheidung muss vielmehr von dem die Löschung ausführenden Anwender oder dem entsprechenden Programm getroffen werden. Ergibt sich die Lösch- und Änderungslogik bereits aus der Logik der Datenbankstruktur, kann aber auch bereits bei der Definition der Tabelle festgelegt werden, wie die Datensätze der Fremdschlüsseltabelle zu behandeln sind, wenn der zugehörige Datensatz der Primärschlüsseltabelle gelöscht oder geändert wird.

Grundsätzlich können vier Möglichkeiten angegeben werden, wie beim Löschen oder Ändern eines Datensatzes der Primärschlüsseltabelle, also hier eines Kunden, vorzugehen ist:

Tabelle 8.1
Aktionen bei der Definition von Fremdschlüsseln

Aktion	DELETE	UPDATE
`NO ACTION` (Standard)	Der Datensatz in der Primärschlüsseltabelle wird nicht gelöscht, stattdessen wird eine Fehlermeldung erzeugt. Damit wird verhindert, dass in der Fremdschlüsseltabelle unkontrolliert Datensätze entstehen, deren Fremdschlüssel zu keinem Datensatz in der Primärschlüsseltabelle gehören.	Der Datensatz in der Primärschlüsseltabelle wird nicht geändert, stattdessen wird eine Fehlermeldung erzeugt. Damit wird verhindert, dass in der Fremdschlüsseltabelle unkontrolliert Datensätze entstehen, deren Fremdschlüssel zu keinem Datensatz in der Primärschlüsseltabelle gehören.
`CASCADE`	Die Löschung erfolgt mit „Dominoeffekt“. Der Datensatz in der Primärschlüsseltabelle wird gelöscht. Zusätzlich werden die Datensätze mit dem entsprechenden Fremdschlüssel in der Fremdschlüsseltabelle ebenfalls gelöscht.	Die Änderung des Primärschlüssels in der Primärschlüsseltabelle wird durchgeführt (sofern diese erlaubt ist). Zusätzlich werden auch die Fremdschlüssel in der Fremdschlüsseltabelle entsprechend geändert. Die Datensätze „passen“ somit wieder zusammen.

Tabelle 8.1 (Forts.) Aktionen bei der Definition von Fremdschlüsseln

Aktion	DELETE	UPDATE
`SET NULL`	Der Datensatz in der Primärschlüsseltabelle wird gelöscht. In den dazugehörigen Datensätzen der Fremdschlüsseltabelle werden die Fremdschlüsselwerte auf `NULL` gesetzt (sofern erlaubt).	Der Datensatz in der Primärschlüsseltabelle wird geändert. In den dazugehörigen Datensätzen der Fremdschlüsseltabelle werden die Fremdschlüsselwerte auf `NULL` gesetzt (sofern erlaubt).
`SET DEFAULT`	Der Datensatz in der Primärschlüsseltabelle wird gelöscht. In den dazugehörigen Datensätzen der Fremdschlüsseltabelle werden die Fremdschlüsselwerte auf den Standardwert gesetzt.	Der Datensatz in der Primärschlüsseltabelle wird geändert. In den dazugehörigen Datensätzen der Fremdschlüsseltabelle werden die Fremdschlüsselwerte auf den Standardwert gesetzt.

In obigem Beispiel würde jetzt das Ändern des Primärschlüssels des Datensatzes mit `mid=1`, `bnr=4711` und `kid=2`, also beispielsweise die Änderung der Kundennummer `kid` auf „2000“, dazu führen, dass der Schlüssel geändert wird. Wegen der Einstellung `ON UPDATE CASCADE` würden die entsprechenden Einstellungen in der Fremdschlüsseltabelle `bestellung` ebenfalls geändert werden. Würde also die `kid` in „2000“ geändert, würde die `kid` in der Tabelle `bestellung` in allen beiden Datensätzen ebenfalls geändert.

Würde der Datensatz der „Ewald Kuhn KG“ in der Kundentabelle gelöscht, würde dies dazu führen, dass der Datensatz tatsächlich gelöscht wird. Wegen der Einstellung `ON DELETE SET NULL` würde in die beiden Datensätze seiner Bestellungen (siehe Abbildung 8.11) als Wert für die beiden Fremdschlüsselfelder jeweils `NULL` eingetragen.

Referenzielle Integrität

Die Angabe der `ON UPDATE`- und `ON DELETE`-Optionen soll Anomalien vermeiden, also die sogenannte referenzielle Integrität sicherstellen. Referenzielle Integrität bedeutet im Wesentlichen, dass keine sinnlosen Fremdschlüsselbeziehungen existieren und keine notwendigen Beziehungen fehlen, die Daten also „zusammenpassen“.

CASCADE

Wesentlich hierfür ist die Option `CASCADE`, die dies weitgehend sicherstellt. Sie führt bei einer Änderung des Primärschlüssels dazu, dass die zugehörigen Werte in den Fremdschlüsseln ebenfalls geändert werden, sodass Fremdschlüsselwert und Primärschlüsselwert wieder zusammenpassen. Bei einer Löschung des Primärschlüsselwertes in einem Datensatz werden die entsprechenden Eintragungen in der Detailtabelle ebenfalls gelöscht. Problematisch ist `CASCADE` insofern, als insbesondere beim Löschen Informationsverluste auftreten können. Werden mehrere Tabellen über Beziehungen mit `CASCADE` miteinander verbunden, so kann der „Dominoeffekt“ erhebliche Auswirkungen haben, da dann über alle so verbundenen Tabellen hinweg gelöscht wird. Zu beachten ist auch, dass keine „Schleifen“ existieren, die wieder zur ursprünglichen Tabelle zurückführen, da so ebenfalls Fehler entstehen würden.

SET DEFAULT

Die Option SET DEFAULT kann sinnvoll genutzt werden, um Datensätze zu sammeln, die durch Änderungs- oder Löschoperationen ihren Bezug zur Primärschlüsseltabelle verloren haben. Diese Datensätze können dann in nachfolgenden Bearbeitungsschritten entsprechend ausgewertet oder bearbeitet werden, um das Problem dieser fehlenden Bezüge aufzulösen.

SET NULL

Die Option SET NULL ist der Option SET DEFAULT ähnlich. In den Fällen, in denen kein DEFAULT-Wert angegeben ist, führen beide auch zu demselben Ergebnis. Abzuwägen ist die Bedeutung des Ergebnisses. NULL bedeutet das Fehlen einer Information. Insofern ist es sinnvoll, NULL einzusetzen, wenn gerade dies ausgedrückt werden soll, wenn also im Fremdschlüssel deutlich werden soll, dass kein zugehöriger Primärschlüsselwert existiert. Es fehlt die Information über die Beziehung, daher fehlt die Fremdschlüsselinformation. Demgegenüber ist der DEFAULT-Wert dann vorzuziehen, wenn ausgedrückt werden soll, dass diese Datensätze ohne Bezug zur Primärschlüsseltabelle in einer Art „Container“ für Sonstige gesammelt werden sollen.

NO ACTION

Der Standardwert NO ACTION, der auch beim kompletten Fehlen einer Angabe verwendet wird, überträgt die Verantwortung für die referenzielle Integrität und die Behandlung der Datensätze schließlich vollkommen dem Anwender beziehungsweise dem Programm, das die Datenbank ändert. Dies ist immer dann sinnvoll, wenn eine solche Behandlung des Problems auch tatsächlich erfolgt. In diesem Fall behindert die Datenbank die Problembehandlung am wenigsten, da keinerlei Automatismen greifen und keinerlei Information ungewollt verloren geht.

Referenzielle Integrität in MS Access

Eine Besonderheit von MS Access ist die Möglichkeit, die Optionen zur referenziellen Integrität wiederum über die Oberfläche bearbeiten zu können. Dazu wird hier von einer Beziehung ohne referenzielle Integrität ausgegangen, wie in Abbildung 8.12 dargestellt.

Jetzt wird die entsprechende Beziehung zunächst mit der Maus markiert. Dann kann sie entweder über die rechte Maustaste oder über BEZIEHUNGEN/BEZIEHUNGEN BEARBEITEN bearbeitet werden. Dabei wird ein Fenster wie in Abbildung 8.12 verwendet. Im oberen Teil sind die Tabellennamen kunden für die Mastertabelle und bestellung für die Detailtabelle angegeben. In der Mastertabelle sind die Primärschlüsselfelder mid und kid angegeben, denen in der Detailtabelle bestellung die gleichnamigen Fremdschlüsselfelder entsprechen. Im unteren Fenster ist die Zuordnung der Felder in der Beziehung angegeben. Sie sehen in dieser Darstellung gut die paarweise Zuordnung von Primärschlüsselfeld und Fremdschlüsselfeld. Die Tabellendarstellung veranschaulicht außerdem, dass mehrere Felder paarweise in die Beziehung einbezogen werden können, was den jeweiligen Feldlisten in der SQL-Anweisung entspricht.

Abbildung 8.12
Beziehung mit referenzieller Integrität

Im unteren Teil des Fensters kann die referenzielle Integrität aktiviert werden. Wird die referenzielle Integrität nicht aktiviert, entspricht dies der Angabe `SET DEFAULT`. MS Access erlaubt also das Löschen und Ändern ohne Kontrolle der Beziehungen und ersetzt die Einträge im Fremdschlüsselfeld durch einen (datentypabhängigen) Standardwert, beispielsweise 0. Wird nur das entsprechende Kontrollfeld MIT REFERENTIELLER INTEGRITÄT aktiviert, stellt MS Access den Standardwert `NO ACTION` ein. Jetzt wird also das Löschen und Ändern von Datensätzen so kontrolliert, dass keine Verletzung der referenziellen Integrität auftreten kann. Es können also keine Datensätze in die Detailtabelle eingefügt werden, denen kein Wert in einem Primärschlüsselfeld der Mastertabelle entspricht. Hier bedeutet das, dass keine Bestellungen eingefügt werden können, deren Kunde nicht in der Kundentabelle steht. Es können auch keine Datensätze von Kunden aus der Kundentabelle gelöscht werden, die noch Bestellungen in der Tabelle `bestellung` aufweisen. Schließlich können weder Primärschlüssel noch Fremdschlüssel geändert werden.

Die beiden zusätzlichen Optionen AKTUALISIERUNGSWEITERGABE und LÖSCHWEITERGABE ermöglichen ein Verhalten, das der Option `CASCADE` entspricht. Hier werden also bei einer Änderung des Primärschlüssels `kid` oder `mid` in `kunden` die entsprechenden Werte in die jeweiligen Felder der Tabelle `bestellung` übernommen, sodass sie nach einer Änderung wieder übereinstimmen. Die Löschweitergabe bewirkt schließlich im Falle des Löschens eines Datensatzes aus der Tabelle `kunden` das Löschen der entsprechenden Datensätze aus der Detailtabelle `bestellung`.

In Normalfall wird nach dem Einschalten der referenziellen Integrität die Darstellung der Beziehung verändert, indem die Enden der Beziehung mit „1“ beziehungsweise „n“ (mit dem Zeichen für unendlich!) markiert werden (siehe Abbildung 8.13). Die Mastertabelle erhält die „1“. MS Access leitet die Darstellung aus der Verwendung des Primärschlüssels ab. Da dieser eindeutig sein muss, wird stets die Seite einer Beziehung, die den Primärschlüssel enthält, mit „1“ markiert.

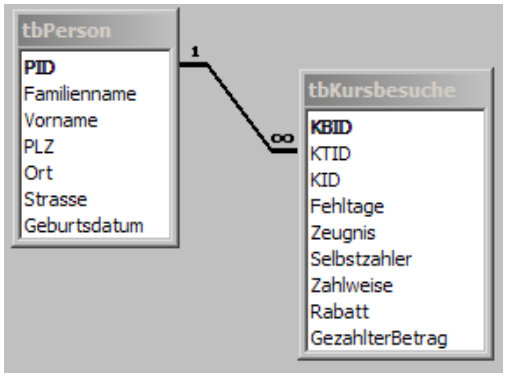


Abbildung 8.13
Beziehung mit referenzieller Integrität

Wenn Sie zwei Tabellen über ihre Primärschlüssel verbinden, erkennt MS Access auch, dass es sich um eine 1:1-Beziehung handelt. Das Verfahren funktioniert leider nicht bei Beziehungen mit mehreren Feldern, sodass im Beispiel der Kunden und deren Bestellungen keine Markierung erfolgt. Der Funktion tut dies aber keinen Abbruch.

8.3.3 Allgemeine Integritätsbedingung (CHECK)

Neben den speziellen auf die Schlüssel bezogenen Bedingungen `PRIMARY KEY` und `FOREIGN KEY` gibt es außerdem die Möglichkeit, allgemeine Integritätsbedingungen anzugeben, die für alle Datensätze der Tabelle eingehalten werden sollen. Mithilfe der `CHECK`-Integritätsbedingung werden diese Bedingungen definiert, die vom Datenbankmanagementsystem dann bei allen Datenänderungen, also insbesondere bei `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen, überprüft werden.

Integritätsbedingung: CHECK

[CONSTRAINT *name]* *CHECK (bedingung1 { AND | OR (bedingung) })*

Der Aufbau der CHECK-Bedingung entspricht im Wesentlichen dem aus der WHERE-Klausel und der HAVING-Klausel der SELECT-Anweisung bekannten Konstrukt. Es können für die Formulierung der Bedingungen alle Literale sowie die Feldnamen der Tabelle verwendet werden. Diese können mit den ebenfalls aus der WHERE-Klausel bekannten Operatoren wie =, <, >, LIKE und anderen zu Bedingungen verknüpft werden. Die Bedingungen lassen sich wiederum mit AND und OR zu komplexen Ausdrücken kombinieren.

CHECK-Bedingung eines Feldes

Die CHECK-Bedingung kann auch direkt an eine Felddefinition angehängt werden:

...

feldname datentyp **CHECK (***bedingung***)**

...

Obwohl beides, solange nur ein Feld betroffen ist, prinzipiell gleichwertig ist, macht es die Analyse einfacher, wenn Bedingungen, die sich nur auf ein Feld beziehen, direkt an die Definition dieses Feldes angehängt werden, während übergreifende Bedingungen als eigene Integritätsbedingungen an das Ende der CREATE TABLE-Anweisung angefügt werden.

Beispiel

Die erweiterte Definition für die Erzeugung der Artikeltabelle zeigt beispielhaft, wie die CHECK-Option sinnvoll genutzt werden kann.

Listing 8.7 Erzeugen der Tabelle artikel mit Integritätsbedingungen

```
CREATE TABLE artikel (
 anr int NOT NULL PRIMARY KEY,
 bezeichnung VARCHAR(255) NOT NULL,
 gebinde FLOAT,
 einheit VARCHAR(30),
 wgknz INT DEFAULT 0 NOT NULL REFERENCES warengruppe(kennziffer)
 ON DELETE CASCADE ON UPDATE CASCADE,
 einstandspreis DECIMAL(6,2) CHECK (einstandspreis >= 0),
 listenpreis DECIMAL(6,2) CHECK(listenpreis >= 0),
 mwst_art CHAR(9) DEFAULT 'voll' NOT NULL REFERENCES mwst(mwst_art)
 ON DELETE SET DEFAULT ON UPDATE CASCADE,
 mindestbestand INT DEFAULT 0 NOT NULL CHECK (mindestbestand >= 0),
 letzte_aenderung timestamp DEFAULT current_timestamp,
 CHECK (listenpreis >= einstandspreis * 1.5));
```

Sie sehen mehrere Einsätze der CHECK-Option. Mit der Angabe

```
einstandspreis decimal(6,2) NULL CHECK (einstandspreis >= 0)
```

wird sichergestellt, dass keine Einstandspreise eingetragen werden können, die kleiner als 0 sind. Derartige Preise machen auch in der Praxis keinen Sinn. Zusätzlich wird als Integritätsbedingung

```
CHECK (listenpreis >= einstandspreis * 1,5)
```

angegeben. Diese CHECK-Bedingung greift auf den *Kalkulationsfaktor* zurück. Dieser wird zur Bestimmung des Listenpreises aus dem Einstandspreis verwendet und beträgt hier 1,5. Damit kann kein Artikel billiger als 50 % über

dem Einstandspreis gelistet werden. Bei der Formulierung der Bedingung werden zwei Felder der Tabelle verwendet und neben dem Vergleichsoperator auch noch eine Rechenoperation eingesetzt. Derartige Operationen werden nicht von allen Datenbanken unterstützt.

Auf Indizes wird später ausführlich eingegangen, weil sie keinen Einfluss auf die logische Struktur haben, sondern der Optimierung von Zugriffszeiten dienen.

Oracle

Oracle kennt in der hier genutzten Version nur die `ON DELETE SET NULL`- und `ON DELETE CASCADE`-Bedingung. Die anderen Varianten und `ON DELETE UPDATE` werden nicht unterstützt.

openBase

openBase verwendet wie beschrieben die `REFERENCES`-Klausel in der aktuellen Version nur als eigene Integritätsbedingung der `CREATE-TABLE`-Anweisung, nicht als Zusatz zur Felddefinition. Daher können auch nur dort die `ON DELETE`- oder `ON UPDATE`-Bedingungen angegeben werden. Die `CHECK`-Bedingung wird ebenfalls nicht direkt bei der Felddefinition, sondern nur als zusätzliche Integritätsbedingung „am Ende“ der `CREATE TABLE`-Anweisung unterstützt. Die entsprechenden `CHECK`-Bedingungen können aber dort einfach ergänzt werden (siehe Listing 8.8).

Listing 8.8
Integritätsbedingungen als eigene Bedingung

```
CREATE TABLE artikel (
  anr INT NOT NULL PRIMARY KEY,
  bezeichnung VARCHAR(255) NOT NULL,
  gebinde FLOAT,
  einheit VARCHAR(30),
  wgknz INT DEFAULT 0 NOT NULL,
  einstandspreis DECIMAL(6,2),
  listenpreis DECIMAL(6,2),
  mwst_art CHAR(9) DEFAULT 'voll' NOT NULL,
  mindestbestand INT DEFAULT 0 NOT NULL,
  letzte_aenderung timestamp DEFAULT current_timestamp,
  CHECK (einstandspreis >= 0),
  CHECK (listenpreis >= 0),
  CHECK (mindestbestand >= 0),
  CHECK (listenpreis >= einstandspreis * 1.5),
  FOREIGN KEY (wgknz) REFERENCES warengruppe(kennziffer) ON DELETE
CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (mwst_art) REFERENCES mwst(mwst_art) ON DELETE SET
DEFAULT ON UPDATE CASCADE
);
```

Die Form der Definition von Integritätsbedingungen, wie in Listing 8.8 angegeben, ist auch in den anderen Datenbankensystemen lauffähig und kann daher generell verwendet werden.

8.3.4 UNIQUE-Bedingung

Als letzte gängige Integritätsbedingung sei hier noch die `UNIQUE`-Bedingung erwähnt. Die Angabe `UNIQUE` erfordert stets die Eindeutigkeit eines Wertes oder einer Gruppe von Werten über alle Datensätze einer Tabelle. Bezieht sich die Eindeutigkeit nur auf ein Datenfeld, so kann der Zusatz `UNIQUE` in der Definition des entsprechenden Feldes erfolgen. Der Zusatz muss dann nach der `DEFAULT`-Angabe stehen, sofern eine solche vorhanden ist. Die meis-

ten Datenbanken unterstützen dies. Damit darf ein Wert in einer mit UNIQUE gekennzeichneten Spalte nur in höchstens einem Datensatz der Tabelle auftreten.

Die Integritätsbedingung UNIQUE kann aber immer als eigene Integritätsbedingung in eine Tabellendefinition aufgenommen werden. Der Zusatz hat dann die Form:

UNIQUE

Integritätsbedingung:

[CONSTRAINT *name]* *UNIQUE (feldname1* { *, feldname* }*)*

Es handelt sich also um eine Aufzählung von Feldnamen, deren Werte gemeinsam eindeutig sein müssen. Das bedeutet, dass eine bestimmte Kombination von Werten in diesen Feldern zusammengenommen nur in höchstens einem Datensatz auftreten darf.

Wird nur ein Feldname angegeben, müssen wiederum die Werte in diesem Feld für jeden Datensatz der Tabelle eindeutig sein. Es handelt sich also bei dem einzelnen Feld beziehungsweise bei jeder Kombination von Feldern immer um Schlüsselkandidaten, wie sie aus der Modellierung heraus entstanden sein können.

Listing 8.9 UNIQUE-Integritätsbedingung

```
CREATE TABLE bestellung (
   mid INT NOT NULL,
   bnr INT NOT NULL,
   kid VARCHAR(50) NOT NULL,
   bestelldatum DATE NOT NULL,
   letzte_aenderung TIMESTAMP DEFAULT current_timestamp,
   CONSTRAINT PKBEST PRIMARY KEY (mid,bnr),
   CONSTRAINT FKBESTKUND FOREIGN KEY (mid,kid) REFERENCES
kunden(mid,kid)
   ON UPDATE CASCADE ON DELETE CASCADE,
   UNIQUE(kid,bestelldatum)
);
```

In Listing 8.9 ist eine Integritätsbedingung eingefügt worden, die festlegt, dass pro Kunde nur einmal am Tag eine Bestellung erfolgen kann.

8.3.5 Übungen zu Integritätsbedingungen

DROP TABLE

Hier sind die Übungen zu den verschiedenen Integritätsbedingungen zusammengefasst. Bitte beachten Sie, dass Sie vorhandene Tabellen löschen müssen, bevor Sie sie neu erzeugen können. Dabei ist auch die Reihenfolge von Bedeutung, wenn Sie mit referenzieller Integrität arbeiten (REFERENCES). Sie können nur Verweise auf Tabellen erzeugen, die bereits existieren. Umgekehrt können Sie nur Tabellen löschen, auf die kein Verweis mehr besteht. Vereinfacht ausgedrückt bedeutet das, dass die Tabellen im Zweifelsfall in umgekehrter Reihenfolge ihrer Erstellung gelöscht werden müssen, wenn sie mit REFERENCES-Klauseln miteinander verbunden sind. Sie löschen Tabellen mit:

DROP TABLE *tabellenname;*

Übungen

1. Erstellen Sie alle Tabellen des Schemas `artikel`, sofern noch nicht geschehen. Sie können dabei auf die Listings des Buches zurückgreifen oder diese ergänzen, wenn Sie bereits erstellte Tabellen noch einmal löschen und mit zusätzlichen Eigenschaften neu erzeugen wollen. (Ü8.3.1)
2. Erstellen Sie eine Tabelle `kind`, die der Speicherung der Kinder der Kunden dient. Die Tabelle soll einen 25-stelligen Vornamen, das Alter in Jahren und das Geschlecht des Kindes beinhalten. Das Alter soll nicht negativ sein. Das Geschlecht soll als „W“ oder „M“ angegeben werden. Andere Angaben sind verboten. Es soll eine Beziehung zur Tabelle `kunden` aufgebaut werden, um das Kind dem Kunden zuordnen zu können. Außerdem sollen das Datum und die Uhrzeit der letzten Änderung gespeichert werden können. (Ü8.3.2)

8.3.6 MS Access

Primärschlüssel in MS Access

MS Access bietet natürlich ebenfalls die Möglichkeit, neue Tabellen anzulegen. Dabei steht aber nur die grafische Oberfläche zur Verfügung. Sie wählen dazu zunächst TABELLEN und dann NEU und schließlich ENTWURFSANSICHT aus. Die Oberfläche sollte dann etwa der in Abbildung 8.14 entsprechen.

Sie können jetzt die einzelnen Felder angeben. Dies entspricht der Felddefinition des `CREATE TABLE`. Zu jedem Feld wird neben dem Namen ein Datentyp angegeben, der den speziellen MS Access-Typen entspricht. Weitere Angaben sind datentypabhängig und werden im unteren Bereich gemacht. So finden Sie für den Datentyp *Text* dort die Längenangabe. Der Standardwert entspricht der `DEFAULT`-Klausel.

Die Angabe EINGABE ERFORDERLICH ist natürlich die Umsetzung der `NULL`- beziehungsweise `NOT NULL`-Angabe in einer `CREATE TABLE`-Anweisung.

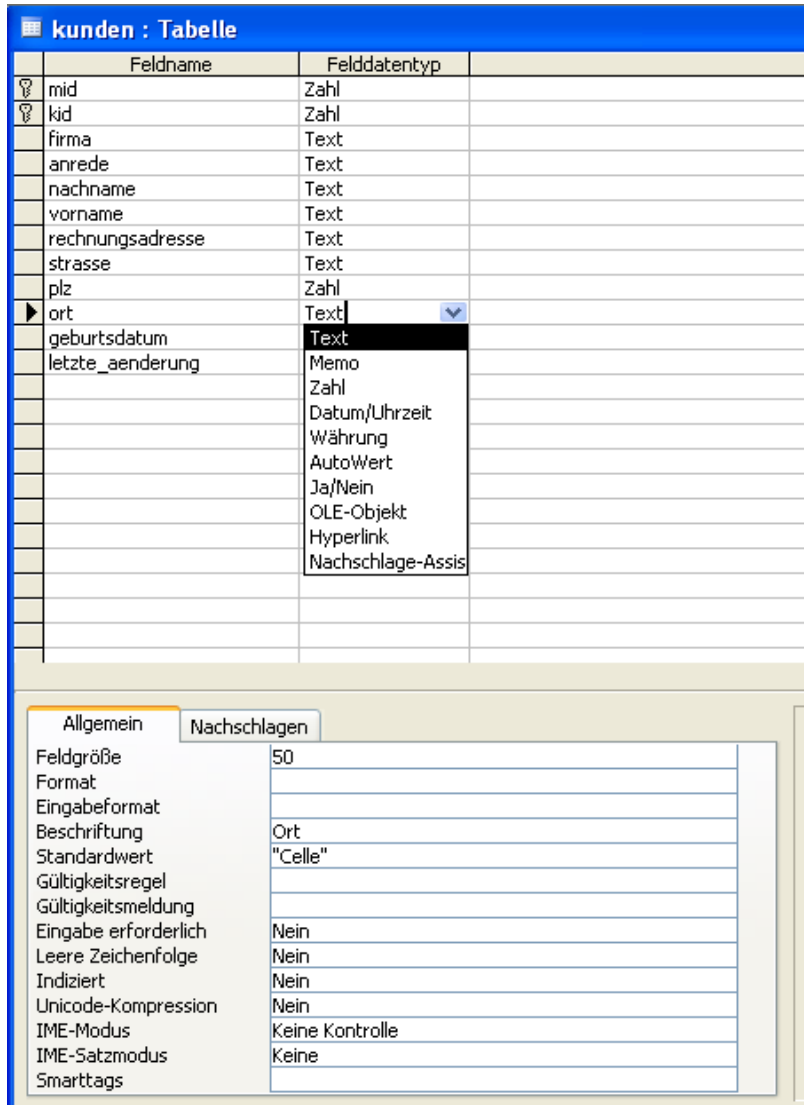
Weitere Angaben wie die GÜLTIGKEITSREGEL oder die Möglichkeit, leere Texte zu unterbinden, weisen bereits auf die `CHECK`-Klauseln hin.

Der Primärschlüssel wird durch Markierung in der linken Spalte vor dem Feldnamen (Mausklick) ausgewählt. Sollen in MS Access mehrere Schlüsselfelder angegeben werden, so können Sie diese Felder über die Entwurfsansicht definieren. Markieren Sie dazu mit gedrückter Steuerungstaste Strg die gewünschten Spalten und definieren Sie sie als Primärschlüssel. Dies geschieht entweder über den Menübefehl BEARBEITEN/PRIMÄRSCHLÜSSEL oder über die entsprechende Schaltfläche. Beachten Sie, dass mit jeder neuen Definition eines Schlüssels der bisherige Schlüssel entfernt wird. Es ist also zwingend notwendig, alle Primärschlüsselfelder gleichzeitig zu markieren. Das Ergebnis sollte wie in Abbildung 8.14 dargestellt aussehen.

Primärschlüssel beim Import

Beachten Sie bitte auch, dass diese Art der Schlüsselvergabe nur in der Entwurfsansicht von MS Access möglich ist. Während des Imports können Sie nur ein Feld als Schlüsselfeld auswählen. Dies führt aber in vielen Fällen zu Datenkonflikten, da die Werte in nur einem Feld nicht eindeutig sind. Daher sollten Sie während des Imports auf eine Primärschlüsselauswahl komplett verzichten.

Abbildung 8.14
Felddefinition in MS Access



Ein Primärschlüssel, der aus mehreren Feldern besteht, erfordert nur, dass die Kombination der Werte aller Primärschlüsselfelder von der Kombination der Werte aller Primärschlüsselfelder aller anderen Datensätze verschieden ist. Einzelne Primärschlüsselfelder dürfen also denselben Wert beinhalten.

Fremdschlüsselbeziehungen (REFERENCES) werden über die Beziehungsansicht erzeugt.

8.4 Die Tabellen ändern (ALTER TABLE)

Wir haben jetzt Tabellen angelegt. Die Datenbank steht. Es können aber immer Situationen entstehen, die Änderungen der Datenbankstruktur erfordern.

Beispiel

Sollen beispielsweise die Artikel in Ihrer Tabelle wegen der Einführung der EAN-Nummern, der sogenannten europäischen Artikelnummern, zusätzlich diese Nummern erhalten, benötigen Sie ein neues Feld für die EAN-Nummer. Jetzt könnten Sie die Tabelle natürlich löschen und sofort wieder neu anlegen. Dabei gehen leider alle Daten verloren. Diese müssten Sie also vorher sichern, dann die Tabelle löschen, die Struktur neu anlegen und die Daten wieder laden. Das ist machbar, aber aufwendig.

Stattdessen können Sie auch eine `ALTER TABLE`-Anweisung verwenden.

Beispiel

```
ALTER TABLE artikel ADD ean VARCHAR(13) NULL;
```

Die `ALTER TABLE`-Anweisung fügt ein neues Feld an die vorhandene Tabelle `artikel` an, hier das Feld `ean`. Die Angaben für die Felddefinition entsprechen den Angaben des `CREATE TABLE`. Die neue Spalte enthält natürlich noch keine Werte, dafür fehlt dann noch ein `UPDATE`. Die `ALTER TABLE`-Anweisung stellt eine elegante Art dar, nachträglich Tabelleninformationen zu ändern.

Dabei können Sie:

- Felder anfügen
- Felder ändern
- Felder löschen
- Integritätsbedingungen anlegen
- Integritätsbedingungen löschen

Tipp

Tatsächlich ist es nicht unüblich die gesamten Fremdschlüssel nicht mit `CREATE TABLE` zu erzeugen, sondern zunächst alle Tabellen einzeln und ohne Beziehungen zueinander, also ohne `REFERENCES`-Klausel, zu erzeugen.

Anschließend werden die Beziehungen durch `ALTER TABLE`-Anweisungen in der Form

ALTER TABLE *fremdschlüsseltabelle*

ADD CONSTRAINT *Fkname*

FOREIGN KEY (*fremdschlüsselfeld*)

REFERENCES *primärschlüsseltabelle* (*primärschlüsselfeld*);

aufgebaut. Der Vorteil liegt darin, dass bei der Reihenfolge der Tabellenerzeugung (`CREATE TABLE`) nicht auf die Beziehungen zwischen den Tabellen geachtet werden muss, weil noch keine Beziehungen erzeugt werden. Die Reihenfolge der Tabellenerstellung spielt also insofern keine Rolle. Folgen dann die `ALTER TABLE`-Anweisungen als zweiter Teil der Datenbankerstellung, sind bereits alle Tabellen vorhanden und es kann bei der Erstellung der Fremdschlüsselbeziehungen mit der `REFERENCES`-Klausel zu keinerlei Konflikten wegen fehlender Tabellen kommen. So können auch „Schleifen“ in den Beziehungen problemlos eingefügt werden.

Wie erwähnt, lassen sich die Änderungen einer Tabelle auch durch deren Löschung mit der DROP TABLE -Anweisung und der anschließenden neuen Erzeugung in veränderter Struktur mit einem neuen CREATE TABLE erreichen. Dies führt allerdings zu einem Verlust der in der Tabelle gespeicherten Daten.

Der Datenverlust lässt sich vermeiden, indem die Daten zunächst exportiert und dann die Tabelle mit einer DROP TABLE-Anweisung gelöscht wird. Dann kann die Tabelle mit einem geänderten CREATE TABLE neu erzeugt werden. Anschließend müssen die Daten mit einem geänderten INSERT wieder eingefügt werden.

Die Philosophien gehen hier in der Praxis auseinander. Administratoren großer Datenbanken bevorzugen häufig das Entladen, Löschen, Ändern der Tabellenstruktur und das anschließende Laden der Daten. Die Entlade- und Lademechanismen sind zumeist bereits implementiert und der frische Datenbestand erlaubt eine performantere Speicherung. Bei kleineren Systemen stehen diese Mechanismen aber nicht immer zur Verfügung, sodass das beschriebene Vorgehen eher umständlich und fehleranfällig ist. In einem laufenden Betrieb kann dann auch ein Zeitproblem hinzukommen. Daher hat die ALTER TABLE -Anweisung durchaus eine erhebliche Bedeutung im Umgang mit relationalen Datenbanken.

Die ALTER TABLE-Anweisung erlaubt grundsätzlich die in der folgenden Syntax angegebenen Änderungen.

Syntax ALTER TABLE

ALTER TABLE *tabellenname*

{ ADD *felddefinition*

| ALTER *felddefinition*

| DROP *felddefinition*

| ADD *Integritätsbedingung*

| DROP *Integritätsbedingung*

};

Tatsächlich haben die Datenbankhersteller, wegen der hohen Bedeutung einer komfortablen Änderbarkeit der Datenbank, hier vergleichsweise viel Aufwand betrieben und zahlreiche Erweiterungen implementiert, die eine Vielzahl von Steuermöglichkeiten bei der Nutzung von ALTER TABLE bereitstellen.

MySQL

An dieser Stelle soll insbesondere auf MySQL eingegangen werden, da es hier einige Besonderheiten gibt. MySQL erlaubt zwar das standardkonforme Einfügen beliebiger Felder und Integritätsbedingungen, differiert aber beim Löschen der Integritätsbedingungen.

So gibt es kein `ALTER TABLE tabelle DROP CONSTRAINT ...`, sondern stattdessen

```
ALTER TABLE tabelle DROP PRIMARY KEY;
ALTER TABLE tabelle DROP FOREIGN KEY name;
ALTER TABLE tabelle DROP INDEX name;
```

Allgemeine `CHECK`-Bedingungen lassen sich so leider nicht entfernen. Dafür gibt es eine Reihe von Erweiterungen, die beispielhaft in der Tabelle 8.2 zusammengestellt sind.

Tabelle 8.2
Einige Erweiterungen der ALTER TABLE-Anweisung in MySQL

Anweisung	Wirkung
`ALTER TABLE` *tabelle* `ADD COLUMN` *name* `FIRST`	Fügt die neue Spalte am Anfang der Tabelle ein.
`ALTER TABLE` *tabelle* `ADD COLUMN` *name* `AFTER` *name2*	Fügt die neue Spalte hinter der Spalte mit dem Namen `name2` ein.
`ALTER TABLE` *tabelle* `ALTER COLUMN` *name* `SET DEFAULT` *wert*	Setzt einen Standardwert für die angegebene Spalte.
`ALTER TABLE` *tabelle* `ALTER COLUMN` *name* `DROP DEFAULT`	Entfernt den Standardwert für die angegebene Spalte.
`ALTER TABLE` *tabelle* `MODIFY COLUMN` *name weitere Angaben*	Damit kann die Felddefinition geändert werden. Der Datentyp, `NULL`|`NOT NULL` und andere Angaben lassen sich so verändern (funktioniert auch in Oracle).
`ATER TABLE` *tabelle* `CHANGE` *alter_name neuer_name weitere Angaben*	Damit können Felder umbenannt werden. Die weiteren Angaben entsprechen dem `MODIFY`.
`ALTER TABLE` *tabelle* `RENAME TO` *neuer_tabellenname*	Die Tabelle wird umbenannt.

Übungen zur ALTER TABLE-Anweisung

Übungen

1. Erstellen Sie eine neue Spalte `lieblingsspeise` mit maximal 50 Zeichen für die Kinder in der Tabelle `kind` aus Ü8.3.2, die freiwillig angegeben werden kann. (Ü8.4.1)
2. Löschen Sie das Feld `lieblingsspeise` wieder aus der Tabelle `kind`. (Ü8.4.2)
3. Löschen Sie den Fremdschlüssel auf den Kunden. (Ü8.4.3)
4. Löschen Sie die Integritätsbedingung, die beim Geschlecht eine Eingabe von „M“ oder „W“ erfordert. (Ü8.4.4)
5. Fügen Sie eine neue Integritätsbedingung ein, die beim Geschlecht eine Eingabe von „M“ oder „F“ erfordert. (Ü8.4.5)

8.5 Tabellen löschen (DROP TABLE)

Sie können Tabellen vollständig, sowohl mit Struktur als auch mit allen Daten, aus der Datenbank entfernen. Soll beispielsweise die Tabelle `artikel` komplett entfernt werden, so reicht dazu:

```
DROP TABLE artikel;
```

Nachdem Sie diesen Befehl eingegeben haben, ist die komplette Tabelle gelöscht. Dabei werden neben der Struktur natürlich auch sämtliche Daten in der Tabelle gelöscht. Sie sind damit vollständig verloren.

DROP TABLE

Die Syntax lautet also:

DROP TABLE *tabellenname;*

Das Datenbankmanagementsystem prüft allerdings vor der Löschung, ob einer Löschung referenzielle Integritäten entgegenstehen. Referenzielle Integritäten werden bekanntlich durch den `CONSTRAINT`

FOREIGN KEY *fremdschlüsselfeld REFERENCES tabellenname primärschlüsselfeld*

im Rahmen der `CREATE TABLE`- oder `ALTER TABLE`-Anweisung erzeugt. Eine solche Integritätsbedingung definiert das mit *fremdschlüsselfeld* angegebene Feld als Fremdschlüssel in der aktuellen Tabelle. Dass bedeutet für die referenzierte Tabelle `tabellenname`, dass sie von unserer Tabelle benötigt wird.

Beispiel für referenzielle Integrität

Wird beispielsweise in der Tabelle `bestellung` ein Fremdschlüssel auf die Tabelle der Kunden definiert, bedeutet das, dass die Kundentabelle benötigt wird, um bei den Bestellungen auf einen Kunden verweisen zu können. Das bedeutet aber auch, dass die Kundentabelle nicht einfach gelöscht werden darf, da dann die Bestellungen sozusagen „in der Luft hängen" würden. Es wäre nicht mehr möglich, die Bestellungen einem Kunden zuzuordnen. Dies muss die Datenbank verhindern. Sie verbietet daher das Löschen der Kundentabelle, solange in einer anderen Tabelle ein Fremdschlüssel definiert ist, der auf diese Tabelle verweist. Es muss dann entweder zunächst die Bestellungstabelle gelöscht werden oder es muss zumindest mit einer `ALTER TABLE`-Anweisung die Fremdschlüsselbeziehung gelöscht werden. In unserer Artikeldatenbank müsste also ein

```
ALTER TABLE bestellung DROP FOREIGN KEY FKBESTKUND
```

für MySQL, beziehungsweise ein

```
ALTER TABLE bestellung DROP CONSTRAINT FKBESTKUND
```

für andere Datenbanksysteme erfolgen.

Sind für die Tabelle in keiner anderen Tabelle Fremdschlüssel definiert, so existieren auch keine referenziellen Integritäten und die Tabelle wird mit ihrem vollständigen Inhalt gelöscht. Ist dagegen mindestens ein Fremdschlüssel definiert, so „zeigen" die Datensätze in dieser anderen Tabelle auf „unsere" Tabelle und wir haben gesagt, dass ein Löschen der Tabelle nicht erfolgen darf. In diesem Fall ist im Gegensatz zur `UPDATE`- oder `DELETE`-

Anweisung auch das Verhalten, das bei der Erstellung des CONSTRAINT angegeben wurde, nicht entscheidend. Ist dort `SET DEFAULT` oder `SET NULL` angegeben, kann in der anderen Tabelle trotzdem kein `DROP` erfolgen.

Übungen zur DROP TABLE-Anweisung

Übungen

1. Löschen Sie die in Ü8.3.2 angelegte Tabelle `kind`. (Ü8.5.1)
2. Erstellen Sie eine neue Tabelle `test` mit einem Feld `testfeld` vom Typ `INT`, das auch Primärschlüssel ist. Erstellen Sie dann eine zweite Tabelle `test2` mit einem Feld `test2pk` und einem Feld `testfeld2`, beide vom Typ `INT`. Das Feld `test2pk` soll Primärschlüssel sein. Fügen Sie dann eine Integritätsbedingung zu `test2` hinzu, die `testfeld2` als Fremdschlüssel zum Feld `testfeld` in der Tabelle `test` definiert. Versuchen Sie anschließend, die Tabelle `test` zu löschen. (Ü8.5.2)
3. Entfernen Sie `test` jetzt richtig. (Ü8.5.3)
4. Entfernen Sie auch `test2`. (Ü8.5.4)
5. Wie hätten sich beide Tabellen auch entfernen lassen? (Ü8.5.5)

8.6 Benutzer und Programmsichten (CREATE VIEW)

Stellen Sie sich vor, Sie sind stets nur an bestimmten Informationen über Ihre Kunden interessiert, der Rest der Datenbank ist für Sie uninteressant. Dies ist eine typische Situation, wenn verschiedene Personen oder verschiedene Abteilungen dieselbe Datenbank nutzen, also das Normalste der Welt in jedem Unternehmen. Dann muss jeder SQL-Befehl, mit dem Sie auf die Datenbank zugreifen, stets alle Informationen ausfiltern, die Sie nicht benötigen, also immer und immer wieder. In unserer Artikeldatenbank könnte dies der Fall sein, wenn Sie für eines der Unternehmen arbeiten, also für einen Mandanten. Dann müssen Sie in allen Abfragen immer wieder die Mandantennummer verwenden, um sicherzustellen, dass Sie nur die Kunden Ihres Unternehmens sehen. Und vielleicht möchte Ihr Chef das ja auch sicherstellen.

Besser wäre es in diesen Fällen, diese Sicht auf die Daten einmalig zu definieren und dann immer wieder darauf zurückzugreifen.

Datensicht

Sie benötigen in solchen Fällen – entsprechend der Idee des 3-Ebenen-Modells – ein externes Schema, also eine Datensicht auf die Datenbank. SQL bietet die Möglichkeit, solche Sichten zu definieren. Sichten heißen in SQL „`VIEW`“. Ein `VIEW` stellt eine Sicht auf die Tabellen einer Datenbank dar. Er wird, ähnlich wie eine `TABLE`, einmalig definiert und kann dann in vielen SQL-Befehlen, insbesondere in `SELECT`, `DELETE`, `UPDATE` und `INSERT`, verwendet werden. Der `VIEW` filtert die Daten, wie Sie sie benötigen, und kann von Ihnen statt der Originaltabellen als Datenquelle verwendet werden. Er tritt also an die Stelle der `TABLE`.

Das hat entscheidende Vorteile für Sie:

- Sie können für Anwendergruppen oder einzelne Anwender eingeschränkte Sichten auf die Datenbank einrichten und so sicherstellen, dass jeder nur die Daten sieht, die er sehen soll oder darf.
- Sie müssen bei Zugriffen mit SELECT nicht immer wieder die für Sie relevanten Daten filtern.
- Sie können Spalten umbenennen und so dem Anwender für ihn vertraute Begriffe anbieten. Dies kann sowohl die Fachsprache als auch eine Fremdsprache betreffen.
- Sie können Berechnungen und/oder Verdichtungen von Daten durchführen und für Abfragen mit SELECT bereitstellen, ohne die Ergebnisse in der Datenbank zu speichern.
- Sie können – mit gewissen Einschränkungen – auch Änderungen der Daten (UPDATE, INSERT, DELETE) in vereinfachter Form durchführen.
- Sie sind nicht von irgendwelchen Änderungen der Tabellen abhängig, die die Datenbankadministration für andere Benutzer durchführt und die Sie nicht betreffen.

Diese Aufgaben sollte ein VIEW also für Sie erledigen. Ein weiterer Vorteil besteht darin, dass der Datenbankadministrator die Verwendung von VIEWs sehr schätzt, erlauben sie ihm doch in gewissem Umfang Änderungen an der Datenbank, ohne dies mit allen Anwendern besprechen zu müssen. Wenn Sie selbst der Datenbankadministrator sind, werden Sie dies spätestens dann nachvollziehen können, wenn Sie bereits eine Reihe von SELECT-Befehlen entwickelt und unter Umständen in andere Anwendungen oder Programme eingebunden haben.

VIEW

Ein VIEW ist eine spezielle Sicht auf die Datenbank, die weitgehend wie eine Tabelle verwendet werden kann. Ein VIEW dient bestimmten Anwendern oder Programmen zum Zugriff auf die Daten in vorgefilterter Form. Die SQL-Syntax zur Erzeugung eines VIEW lautet:

CREATE VIEW *viewname* ***[(feldname, feldname, ...)]***
AS

SELECT *Select-Klausel;*

Beispiel

Eine Sicht auf die Kunden des Mandanten 1, die BüroFix KG, kann als VIEW auf die Tabelle Kunden definiert werden (siehe Listing 8.10).

Listing 8.10 VIEW auf die Kunden der BüroFix KG

```
CREATE VIEW kundenBueroFix
AS
SELECT * FROM kunden WHERE mid=1;
```

Dabei wird vorausgesetzt, dass alle Kunden der BüroFix KG die Mandantennummer (mid) 1 haben.

Ist ein VIEW einmal definiert, kann er mit SQL-Befehlen (mit gewissen Einschränkungen) wie eine Tabelle bearbeitet werden. Im Gegensatz zu einer Tabelle enthält ein VIEW aber keine physikalischen Daten, sondern stellt die

Daten selbst mithilfe einer SELECT-Anweisung bereit. Es gibt Einschränkungen, wenn Daten in einem VIEW mit INSERT, UPDATE oder DELETE verändert werden sollen.

Prinzipiell kann ein VIEW drei verschiedene (auch kombinierbare) Aufgaben erledigen:

1. Auswahl von Feldern (Spaltenselektion)
2. Auswahl von Datensätzen (Zeilenselektion)
3. Kombination von Tabellen (JOIN)

8.6.1 Spaltenselektion

Wir wollen für die weiteren Beispiele zunächst wieder auf die Kursdatenbank zurückgreifen, da Sie diese bereits von der Vorstellung der SELECT-Anweisung kennen. Stellen Sie sich vor, dass Sie einen Brief an die Kursteilnehmer und Dozenten schreiben wollen, um eine Terminänderung mitzuteilen. Sie wollen die Beteiligten einfach nur anschreiben. Sie brauchen also den Namen und die Adresse. Sie benötigen nicht die Kundennummer, das Geburtsdatum oder andere Informationen.

Info

Erfahrene Datenbankanwender werden hier einwenden, dass die Kundennummer der einzige Schlüssel ist und daher für einen Zugriff verwendet werden sollte. Unsere Sachbearbeiter bevorzugen aber, sich nur Namen zu merken, mit den Nummern hat es schon zu viele Zahlendreher gegeben. Ist dieser einmal nicht eindeutig, fragen sie lieber nach.

Beispiel: VIEW Zeilenselektion

Die Lösung mit einem SELECT kann dann wie folgt aussehen:

Listing 8.11
Namen und Adressen aus der Tabelle tbPerson

```
SELECT
   Familienname,
   Vorname,
   PLZ,
   Ort,
   Strasse
FROM tbPerson;
```

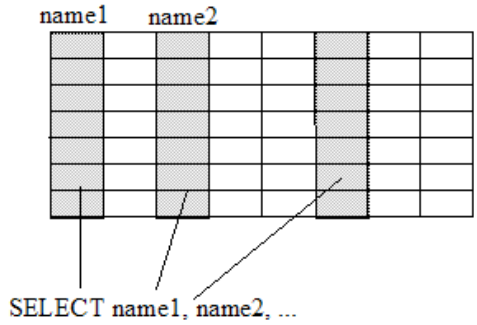


Abbildung 8.15
VIEW als Auswahl von Feldern aus einer Tabelle (Spaltenselektion)

Mit dem SELECT werden die gewünschten Felder für eine Datenabfrage ausgewählt. Schematisch entspricht das der Darstellung in Abbildung 8.15. Sie erhalten genau die Spalten, die grau dargestellt sind. Nehmen Sie den SELECT-Befehl jetzt als Datenquelle eines VIEW und fügen ihn in eine CREATE VIEW-Anweisung ein, ergibt sich das Listing 8.12.

Listing 8.12 Definition eines VIEW mithilfe einer SELECT-Anweisung

```
CREATE VIEW postanschrift AS
SELECT
   Familienname,
   Vorname,
   PLZ,
   Ort,
   Strasse
FROM tbPerson;
```

Damit ist eine Sicht, ein VIEW, mit dem Namen postanschrift auf die Tabelle tbPerson festgelegt. Basierend auf diesem VIEW kann beispielsweise Herr Peter Weiss – wie auch alle anderen Personen – künftig mit einem

```
SELECT *
FROM postanschrift
WHERE Familienname='Weiss'
AND Vorname='Peter';
```

gefunden werden. Wird die Tabelle tbPerson künftig um weitere Spalten erweitert, beispielsweise die Umsatzsteuer-ID für Geschäftskunden, Hinweise auf Vegetarier und Veganer oder andere Angaben, so muss das unseren Postversand nicht stören. Er wird in seinem VIEW davon nichts mitbekommen. Wird die Tabelle von tbPerson in tbKunde umbenannt, so ist nur der VIEW einmalig zu ändern, alle darauf basierenden SQL-Anweisungen funktionieren danach wie zuvor.

8.6.2 Zeilenselektion

Ein VIEW erlaubt es nicht nur, bestimmte Spalten aus einer Tabelle auszuwählen, sondern auch bestimmte Datensätze zu selektieren. Sie wollen beispielsweise nur diejenigen anschreiben, die in Braunschweig wohnen? Sie wollen ihnen besondere Angebote, Specials oder ein Gewinnspiel anbieten? Sie wollen sie gezielt anschreiben? Dann wollen Sie vielleicht nicht immer wieder das

```
WHERE Ort='Braunschweig'
```

Beispiel: VIEW Zeilenselektion

in Ihre SQL-Anweisungen einfügen. Sie wollen auch sicher sein, dass nur die in Braunschweig wohnenden Personen angeschrieben werden. In diesem Fall erstellen Sie einen VIEW:

Listing 8.13 Definition eines VIEW mit Selektion

```
CREATE VIEW braunschweig AS
SELECT *
FROM tbPerson
WHERE Ort='Braunschweig';
```

Mit diesem Befehl erzeugen Sie eine Sicht mit dem Namen braunschweig, die ausschließlich die Datensätze beinhaltet, bei denen das Feld Ort den Wert „Braunschweig“ aufweist. Es wird natürlich keine echte Kopie erstellt, sondern beim Zugriff auf den VIEW werden die betreffenden Datensätze aus der

zugrunde liegenden Tabelle `tbPerson` mit der angegebenen `WHERE`-Klausel herausgefiltert. Der Unterschied zu der Verwendung einer `WHERE`-Klausel im `SELECT` liegt lediglich darin, dass Sie sicher sein können, dass sie immer und in jeder `SELECT`-Anweisung ausgeführt wird, die auf dem `VIEW braunschweig` beruht.

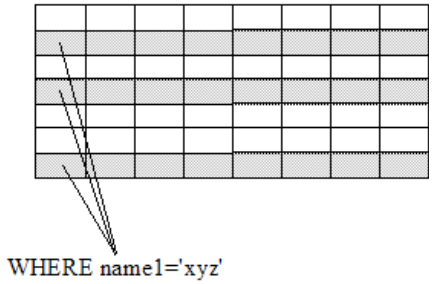


Abbildung 8.16
In dem VIEW sind nur noch die angegrauten Datensätze sichtbar.

Jetzt können Sie mit einem einfachen

```
SELECT * from braunschweig;
```

alle Datensätze der Personen aus Braunschweig, also alle in Abbildung 8.16 angegrauten Zeilen – und nur diese –, erhalten. Natürlich können Sie den `SELECT`-Befehl zur Definition des `VIEW` mit allen in Kapitel 4 beschriebenen Optionen erweitern.

8.6.3 Tabellen kombinieren

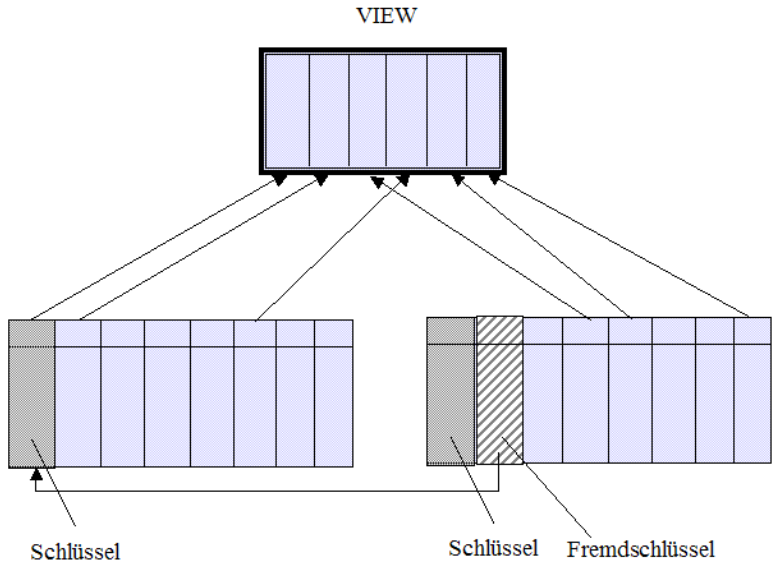
Eine zentrale Bedeutung besitzt der `VIEW` auch, um die Informationen aus mehreren Tabellen zusammenzufassen und dem SQL-Anwender als eine „Tabelle“ zu präsentieren. Dabei können aus zwei oder mehr Tabellen die gewünschten Spalten (und Zeilen) gewählt werden. Für den Anwender hat dies den Vorteil, dass er sich keine Gedanken über die Struktur der Datenbank machen muss, also darüber, welche Information in welcher Tabelle steht. Die Fremdschlüsselbeziehungen, verschiedene Arten des `JOIN` und die richtige Kombination entsprechender Anweisungen müssen nicht immer wieder neu konzipiert werden. Die Entwicklung erfolgt einmalig und anschließend kann auf den `VIEW` statt auf die einzelnen Tabellen zugegriffen werden.

Natürlich muss die Datenbank „wissen“, wie die Daten aus den verschiedenen Tabellen kombiniert werden sollen. Daher muss in dem `VIEW` über entsprechende Fremdschlüssel, also einen `JOIN`, beispielsweise mit

```
WHERE Tabelle1.Schlüssel = Tabelle2.Fremdschlüssel
```

angegeben werden, wie die Zeilen der Tabellen kombiniert werden sollen. Sie erinnern sich, das ist die „alte“ SQL-Syntax. Natürlich können Sie auch mit dem neuen `INNER JOIN`, `OUTER JOIN` oder einem `NATURAL JOIN` arbeiten.

Abbildung 8.17
Kombination zweier Tabellen in einem VIEW



Beispiel: VIEW-Tabellenkombination

Sollen beispielsweise die Adressen aller Kursteilnehmer des Kurses „CE23“ angegeben werden, können die Namen und Adressdaten der Personentabelle über einen `JOIN` gefiltert mit der Tabelle der Kursbesuche als `VIEW` bereitgestellt werden (siehe Listing 8.14).

Listing 8.14
VIEW mit JOIN über zwei Tabellen

```
CREATE VIEW KursCE23 AS
SELECT
   p.Familienname, p.Vorname, p.PLZ, p.Ort, p.Strasse
FROM tbPerson p INNER JOIN tbKursbesuche kb ON (p.PID = kb.KTID)
WHERE kb.KID = 'CE23'
ORDER BY p.Familienname ASC;
```

Auf diesen `VIEW` kann dann zugegriffen werden:

```
SELECT * FROM KursCE23;
```

Das Ergebnis kann wie in Abbildung 8.18 angegeben aussehen.

Abbildung 8.18
Ergebnis des SELECT auf den VIEW KursCE23

Familienname	Vorname	PLZ	Ort	Strasse
Karmann	Thomas	29227	Celle	Trift 28
Klötzer	Karl	29221	Celle	Bahnhofstraße 2
Magerkurth	Melissa	29336	Nienhagen	Am Tümpel 3
Martens	Melanie	29221	Celle	Horstweg 258
Müller	Claudia	29596	Breitenhees	In den Fuhren 12
Ruppert	Nicola	29301	Bergen	Welfenallee 23
Schulze	Tanja	29308	Winsen	Berliner Ring 23
Winter	Petra	29320	Hermannsburg	Immenhoop 51

Der `VIEW` selbst wird zu einem eigenen Objekt in der Datenbank. In den einzelnen Systemen wird er unterschiedlich dargestellt. Die Abbildung 8.19 zeigt die Darstellung in MySQL. Denken Sie an die rechte Maustaste und die Aktualisierung.

Abbildung 8.19
Der VIEW kursce23 ist in das Schema kurse aufgenommen worden.

Achtung

Wichtig für die Eindeutigkeit des VIEW und die Vermeidung von kartesischen Produkten (CROSS JOIN) ist die Angabe des JOIN (also beispielsweise p.PID = kb.KTID), der sicherstellt, dass die beiden Tabellen tbPerson und tbKursbesuche über den Schlüssel beziehungsweise Fremdschlüssel auch im VIEW wirklich miteinander verbunden werden. Neben den Spalten, die Teil des VIEW sind, müssen auch die nur über die WHERE-Klausel oder anderen Teile der SELECT-Anweisung beteiligten Tabellen angegeben werden.

Probleme ergeben sich regelmäßig, wenn Spalten mit identischen Namen existieren. Diese Spalten müssen daher insbesondere in einem VIEW durch Qualifikation mit dem Tabellennamen unterschieden werden, um eine Eindeutigkeit herzustellen. Das Konzept, die Tabellen dafür mit Variablen, p, kb oder anderen sinnvollen Namen, innerhalb der SQL-Anweisung zu benennen, kennen Sie bereits aus Kapitel 4. Sie sollten es hier konsequent verwenden. Zum einen erhöht es die Lesbarkeit des VIEW, zum anderen sichert es auch bei späteren Datenbankänderungen die Eindeutigkeit des VIEW.

8.6.4 Der VIEW in MySQL

Das Konzept der Datensichten (VIEW) ist in relationalen Datenbankmanagementsystemen für den unternehmensweiten Einsatz wie Oracle oder DB2 seit Langem verfügbar. MySQL kennt seit der Version 5 eine richtige VIEW-Anweisung. Daher soll hier exemplarisch am Beispiel von MySQL die Vielfalt der Erweiterungen gezeigt werden, die bei einem CREATE VIEW umgesetzt werden können. Die vollständige Syntax in MySQL lautet:

CREATE VIEW MySQL

CREATE [OR REPLACE]

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]

VIEW *viewname [(feldname, feldname, ...)]*

AS *Select-Anweisung*

[WITH [CASCADED | LOCAL] CHECK OPTION]

Die Angabe eines REPLACE führt dazu, dass ein bereits vorhandener VIEW durch den neuen VIEW ersetzt wird. Anderenfalls muss ein VIEW vor der erneuten Erzeugung gelöscht werden (Sie ahnen es bereits: mit DROP VIEW viewname).

Eine Besonderheit von MySQL ist die Option ALGORITHM. Die Angabe MERGE führt dazu, dass der SQL-Interpreter versucht die Angaben des CREATE VIEW mit den Angaben einer SELECT-Anweisung zu vermischen, mit der wiederum auf den VIEW zugegriffen wird. Nehmen wir an, dass Sie in der Kursdatenbank einen weiteren VIEW TeilnehmerZahlungenCE23 erzeugt haben, wie in Listing 8.15 angegeben.

Listing 8.15 Erzeugung eines VIEW mit MERGE-Option

```
CREATE
   ALGORITHM = MERGE
VIEW TeilnehmerZahlungenCE23 (TeilnehmerID, Zahlung)
AS
SELECT KTID, GezahlterBetrag
FROM tbKursbesuche
WHERE KID = 'CE23';
```

Beachten Sie bitte auch, dass in der VIEW-Definition die Felder umbenannt worden sind. Durch die Angabe der Feldnamen nach dem Namen des VIEW werden neue Feldnamen definiert, denen die von der SELECT-Anweisung gelieferten Felder in gleicher Reihenfolge zugeordnet werden.

Der VIEW kann bekanntlich wie jede Tabelle verwendet werden. Sie können daher darauf auch mit einem SELECT mit einer weiteren WHERE-Klausel zugreifen, wie in Listing 8.16 zu sehen.

Listing 8.16 SELECT-Anweisung auf Basis eines VIEW

```
SELECT *
FROM TeilnehmerZahlungenCE23
WHERE Zahlung < 350;
```

MERGE

Die Angabe der Option ALGORITHM=MERGE bewirkt, dass der SQL-Interpreter von MySQL jetzt die beiden Anweisungen zu einer kombinierten Anweisung mischt:

Listing 8.17 Die (virtuelle) SELECT-Anweisung aus der Kombination von VIEW und SELECT

```
SELECT KTID AS "TeilnehmerID", GezahlterBetrag AS "Zahlung"
FROM tbKursbesuche
WHERE (KID = 'CE23') AND (GezahlterBetrag < 350);
```

Der * in der SELECT-Anweisung wird dabei zunächst in die Felder TeilnehmerID und Zahlung im Sinne des VIEW umgesetzt. Diesen beiden Feldnamen im VIEW entsprechen laut der Definition des VIEW die Feldnamen KTID und GezahlterBetrag der Tabelle tbKursbesuche. Entsprechende Alias werden zusätzlich generiert, um die gemäß VIEW definierten Feldnamen bereitzustellen.

In der WHERE-Klausel werden die Bedingung KID='CE23' des VIEW und die Bedingung Zahlung < 350 aus der SELECT-Anweisung zusammengefügt. In diesen Bedingungen werden die Feldnamen aus der Tabelle verwendet. Der Feldname Zahlung aus dem SELECT des VIEW wird dabei noch durch den entsprechenden Feldnamen GezahlterBetrag der Tabelle tbKursbesuche ersetzt. Beide Bedingungen müssen erfüllt werden, sodass sie durch ein AND miteinander verbunden werden.

TEMPTABLE

Dieser MERGE ist zumeist die performanteste Art einen VIEW zu realisieren, da der SQL-Interpreter eine gemeinsame Optimierung der Abfrage vornehmen kann. Die Alternative zu einem MERGE ist die sequenzielle Ausführung beider Anweisungen, die mit der Angabe ALGORITHM=TEMPTABLE veranlasst wird. Zunächst wird dabei die SELECT-Anweisung des VIEW ausgeführt und das Ergebnis in einer temporären Tabelle (TEMPTABLE) gespeichert. In diesem Fall würde eine Tabelle mit zwei Spalten und den Datensätzen des Kurses „CE23“ erzeugt. Basierend auf dieser Tabelle wird dann die eigentliche SELECT-Anweisung ausgeführt und das endgültige Ergebnis erzeugt.

UNDEFINED

Die dritte mögliche Angabe für die Option ALGORITHM ist UNDEFINED. Dabei wird MySQL die Entscheidung darüber überlassen, wie die Ausführung erfolgen soll. Dies ist zugleich der Standardwert, der immer dann genommen wird, wenn keine andere Angabe erfolgt. In den meisten Fällen wird MySQL dann einen MERGE erzeugen. Es gibt allerdings auch eine Reihe von Fällen, in denen dies nicht möglich ist, insbesondere immer dann, wenn der VIEW die Anzahl der Datensätze ändert, beispielsweise durch ein DISTINCT, ein UNION (siehe hierzu Kapitel 11) oder eine Aggregatfunktion, wie eine Summierung oder Mittelwertbildung in Verbindung mit einer GROUP BY-Klausel.

8.6.5 Der VIEW in MS Access

MS Access kennt leider keine direkte CREATE VIEW-Anweisung. Eine Datensicht entsteht in MS Access, indem Sie eine normale Abfrage erstellen. Jedes Abfrageobjekt wird in MS Access wie ein VIEW behandelt. Jede Abfrage ist ein VIEW. Das heißt insbesondere, dass jede Abfrage wieder die Basis für eine weitere Abfrage sein kann. Sie steht gleichberechtigt neben der Tabelle. Die Abfrage wird als SELECT-Klausel in eine andere Abfrage eingefügt. Das erkennen Sie auch an der Grundstruktur der Oberfläche von MS Access. Die Abbildung 8.20 zeigt das typische Auswahlfenster, mit dem Tabellen und Abfragen ausgewählt werden, die dann als Basis einer Abfrage (also einer SELECT-Anweisung) verwendet werden können.

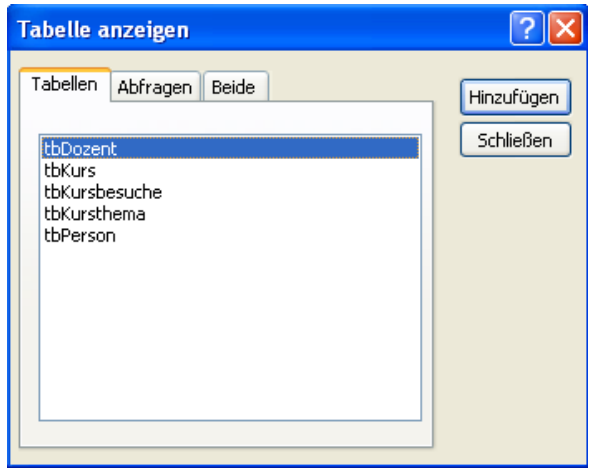


Abbildung 8.20
Auswahl für eine Abfrage in MS Access

Sie sehen unmittelbar, dass beide, Tabellen und Abfragen, gleichberechtigt nebeneinanderstehen. Sie können entsprechend auch in identischer Weise als Basis für eine neue Abfrage verwendet werden.

8.6.6 Einen VIEW ändern (ALTER VIEW, DROP VIEW)

ALTER VIEW

Ist ein VIEW einmal definiert, kann er mit einem ALTER VIEW geändert werden. Der ALTER VIEW entspricht in seiner Syntax weitgehend dem CREATE VIEW.

ALTER VIEW *viewname [(feldname, feldname, ...)]*

AS *Select-Anweisung;*

Ein VIEW speichert keine physischen Daten. Somit könnten Sie einwenden, dass man den VIEW einfach löschen und neu erzeugen könnte, ein ALTER VIEW somit überflüssig ist. Der ALTER VIEW hat aber gegenüber dem ebenfalls möglichen Löschen eines VIEW mit

DROP VIEW

DROP VIEW *viewname;*

und anschließendem erneuten CREATE VIEW den Vorteil, dass eventuelle Berechtigungen und andere Eigenschaften des VIEW erhalten bleiben. Ein VIEW wird gezielt für bestimmte Benutzer oder Benutzergruppen geschaffen. Die damit verbundenen Einstellungen durch die Datenbankadministratoren können einigen Aufwand erfordern. Es ist auch ärgerlich, wenn Programme, die auf einen VIEW zugreifen, beispielsweise in einem Batch-Lauf abbrechen, weil der VIEW keine entsprechenden Berechtigungen mehr enthält.

Der ALTER VIEW ist in dieser Form nur in MySQL vollständig implementiert. Oracle bietet eine Reihe von anderen Änderungsmöglichkeiten.

8.6.7 Änderbarkeit eines VIEW

Grundsätzlich gilt, dass ein VIEW, der einmal definiert ist, überall verwendet werden kann, wo in einer der folgenden, bereits besprochenen SQL-Anweisungen eine Tabelle zulässig ist:

- SELECT
- DELETE
- INSERT
- UPDATE
- sowie einigen weiteren wie COMMENT oder LOCK

Updateable

Leider bringt die Verwendung eines VIEW auch Einschränkungen mit sich. Diese betreffen hauptsächlich die Änderung von Daten in der Datenbank, also insbesondere die SQL-Befehle INSERT, UPDATE und DELETE. Je nachdem, ob die Daten eines VIEW mit diesen Anweisungen änderbar sind oder nicht, spricht man von einem änderbaren (updateable) oder einem nicht änderbaren (non-updateable) VIEW. Nur zur Klarstellung: Es geht nicht um die Änderbarkeit des VIEW, sondern um die Änderbarkeit der Daten in der Datenbank über den VIEW, also beispielsweise ein UPDATE viewname ...

Ein VIEW, der nicht updateable ist, wird häufig auch als Read-only-VIEW bezeichnet.

Grundsätzlich entstehen immer dann Probleme bei der Änderbarkeit von Daten über einen VIEW, wenn sich aus den Datensätzen eines VIEW nicht eindeutig die Datensätze der dem VIEW zugrunde liegenden Tabellen ableiten lassen. Schließlich müssen die richtigen Datensätze in den Tabellen gefunden werden können, um sie ändern oder löschen zu können. Sind die Datensätze nicht eindeutig identifizierbar, so kann in einem VIEW kein DELETE oder UPDATE durchgeführt werden.

Abhängig von der Anzahl der dem VIEW zugrunde liegenden Tabellen gibt es verschiedene mögliche Gründe, warum ein VIEW nicht änderbar ist:

- Der VIEW bezieht sich auf mehrere Tabellen, insbesondere auf Tabellen mit einer 1:n-Beziehung.
- Der VIEW enthält nur Literale und bezieht sich somit auf überhaupt keine Tabelle.
- Der VIEW bezieht sich auf nur eine Tabelle, aber ein Datensatz des VIEW entspricht nicht 1:1 einem Datensatz der zugrunde liegenden Tabelle, es fehlt insbesondere der Primärschlüssel oder Teile des Primärschlüssels.

Mehr als problematisch sind also zumeist VIEWs, die sich nicht auf genau eine Tabelle beziehen. Es gibt aber auch verschiedene Gründe, warum ein VIEW, der auf nur einer Tabelle basiert, nicht änderbar ist. Insbesondere erhalten Sie auch keinen änderbaren VIEW, wenn ...

- Aggregatfunktionen wie SUM, AVG, MIN oder MAX verwendet werden,
- eine GROUP BY-Klausel verwendet wird,
- eine HAVING-Klausel verwendet wird,
- ein DISTINCT verwendet wird,
- ein UNION (siehe Kapitel 11) verwendet wird,
- Unterabfragen (siehe Kapitel 10) verwendet werden.

Sie sollten bei der Definition eines VIEW also berücksichtigen, ob Sie ihn nur für Auswertungen (SELECT) oder auch für Veränderungen verwenden wollen (UPDATE, INSERT, DELETE). Da die Updatefähigkeit eines VIEW sehr oft nicht gegeben ist, muss hier bei der Definition sehr vorsichtig vorgegangen werden. Im Übrigen kann die Änderbarkeit sofort nach der Definition eines VIEW von der Datenbank analysiert werden. Einige Datenbankadministrationswerkzeuge zeigen diese Updatefähigkeit unmittelbar an.

Die Einschränkungen in der Updatefähigkeit resultieren letztlich immer daraus, dass die Datenbank die SELECT-Anweisung zwar zur Gewinnung, Verdichtung und Kombination von Daten bei der Darstellung nutzen kann, sie aber umgekehrt bei einer Änderung nicht mehr auf die einzelnen Felder einzelner Datensätze zurückführen kann.

CHECK OPTION

Um das Konzept der Benutzersicht umsetzen zu können und maximalen Komfort für Abfragen sicherzustellen, werden neben den normalen VIEWs daher normalerweise eigene VIEWs für das Update erstellt. Eine Besonderheit, die aber die Kontrolle über die durch einen Benutzer geänderten Daten noch

einmal verbessert, ist die Prüfoption (CHECK OPTION). Ein VIEW mit CHECK OPTION erlaubt nur die Bearbeitung von Datensätzen, die die Bedingungen der WHERE-Klausel erfüllen.

Betrachten wir dazu noch einmal den zuvor definierten VIEW zur Fokussierung auf die Kunden aus Braunschweig und ergänzen ihn mit der Prüfoption:

```
CREATE VIEW braunschweig AS
SELECT *
FROM tbPerson
WHERE Ort='Braunschweig'
WITH CHECK OPTION;
```

Damit dürfen nur noch Kunden eingefügt oder geändert werden, die aus Braunschweig stammen. Ein Versuch mit

```
INSERT INTO braunschweig
   (PID,Familienname,Vorname,PLZ,Ort,Geburtsdatum)
VALUES
   (99,'Müller','Max',30529,'Hannover','1980-02-01');
```

also der Versuch einen Kunden einzufügen, der nicht aus Braunschweig ist, würde von der Datenbank abgewiesen. Damit ist sichergestellt, dass ein Benutzer nur die ihm gemäß seiner Datensicht zugeordneten Datensätze ändern kann.

Weitere Anwendungen dieser Technik sind viele denkbar, indem beispielsweise nur Kunden eines bestimmten Postleitzahlgebietes oder Aufträge bestimmter Kundengruppen bearbeitet werden können.

8.6.8 Übungen

Übungen

1. Erstellen Sie einen VIEW Geburtstagsliste, der alle Informationen über Personen enthält, die für eine Geburtstagsliste notwendig sind. Achten Sie auf die Sortierung. Testen Sie Ihren VIEW mit einer SELECT-Anweisung. (Ü8.6.1)
2. Erstellen Sie einen VIEW CellerKurse, der die KID, die Kurskennung sowie Kursbeginn und Kursende aller Kurse beinhaltet, die in Celle stattfinden. (Ü8.6.2)
3. Erstellen Sie einen VIEW AccessKurse, der das Kursthema, die Kursbeschreibung sowie die Kurskennung und den Kursbeginn, das Kursende und den Kursleiter mit ID angibt. Sortieren Sie die Liste nach Anfangsdatum und lassen Sie in allen Feldern, die „Kurs“ enthalten, den Teil „Kurs“ im Namen des Feldes weg. (Ü8.6.3)
4. Ändern Sie den VIEW AccessKurse so, dass statt der ID des Kursleiters dessen Name mit in dem VIEW enthalten ist. (Ü8.6.4)
5. Entfernen Sie alle angelegten VIEWs. (Ü8.6.5)
6. Warum führt ein DROP VIEW-Befehl nicht zum Löschen der Daten in dem VIEW? (Ü8.6.6)

8.7 Domänen

8.7.1 Domänen erstellen

SQL kennt Standarddatentypen für ganze Zahlen, Gleitkommazahlen, alphanumerische Werte, Datums- und Zeitangaben. In Kapitel 5 haben wir gesehen, wie die verschiedenen Datenbanksysteme diese umgesetzt und mit verschiedenen Varianten erweitert haben. Trotzdem treten Situationen auf, bei denen nur spezielle Werte für ein Feld Sinn machen. Diese sollten dann nach Möglichkeit auch gleich von der Datenbank geprüft werden.

Info

Domänen sind in der hier beschriebenen Form bei den hier betrachteten Datenbanksystemen bisher nur in Firebird implementiert. Oracle kennt ein Konstrukt zur Definition von Objekten, das aber sehr komplex und Oracle-spezifisch ist und daher hier nicht näher beschrieben werden soll.

Beispiel

Betrachten wir dazu noch einmal die Definition der Tabelle für die Mehrwertsteuer. Das Feld für die Mehrwertsteuerart soll entsprechend der Definition nur die Werte „voll“, „ermässigt“ und „kein“ enthalten können. Demgemäß wurden die Werte in der Tabellendefinition angegeben:

```
CREATE TABLE mwst(
mwst_art ENUM('voll','ermässigt','kein') NOT NULL PRIMARY KEY,
mwst_satz SMALLINT NOT NULL DEFAULT 0
);
```

Eine Alternative ist die Verwendung einer `CHECK`-Integritätsbedingung, wie wir sie für die Datenbanken außer MySQL verwendet haben.

```
CREATE TABLE mwst(
mwst_art char(9)
   CHECK (
      VALUE IN ('voll','ermässigt','kein')
   ) NOT NULL PRIMARY KEY,
mwst_satz SMALLINT NOT NULL DEFAULT 0
);
```

Beide Varianten haben einen gemeinsamen Nachteil: Sie erfordern die Wiederholung der Wertebereich-Definition an allen Stellen, an denen der Mehrwertsteuersatz verwendet wird. Da es sich hier auch noch um den Primärschlüssel handelt, muss die entsprechende Logik an allen Stellen wiederholt werden, an denen ein Fremdschlüsselfeld mit Bezug zur Mehrwertsteuertabelle benötigt wird. Sie erkennen dies am Beispiel der Tabelle `artikel`.

```
CREATE TABLE IF NOT EXISTS artikel (
...
mwst_art ENUM('voll','ermässigt','kein') NOT NULL DEFAULT 'voll' REF-
ERENCES mwst(mwst_art),
...
);
```

Domäne

Das ist nicht nur aufwendig, sondern auch fehlerträchtig. Da sich gerade in der betrieblichen Praxis viele Beispiele für eingeschränkte Wertebereiche finden, hat man nach Wegen gesucht, derartige Spezialwerte und Bedingun-

gen einmalig zu definieren und sie dann an allen benötigten Stellen verwenden zu können. Derartige eingeschränkte Wertebereiche nennt man *Domänen.*

Domänen werden mit einer eigenen CREATE-Anweisung erzeugt, die allerdings nur in wenigen Datenbanksystemen (hier Firebird) verfügbar ist. In unserem Beispiel könnte das wie folgt aussehen:

```
CREATE DOMAIN mehrwertsteuerart
   AS CHAR(9) DEFAULT 'kein' NOT NULL
CHECK(
   VALUE IN ('voll','ermässigt','kein')
);
```

Die Definition der Domäne wird also mit einer eigenen SQL-Anweisung erzeugt. Die Domäne erhält damit auch einen eigenen Namen, hier wird `mehrwertsteuerart` erzeugt. Der Name muss wiederum innerhalb des Schemas eindeutig sein. Er kann als Datentyp in anderen DDL-Anweisungen genutzt werden. Die Tabelle `mwst` kann jetzt erstellt werden mit:

```
CREATE TABLE mwst(
mwst_art mehrwertsteuerart PRIMARY KEY,
mwst_satz SMALLINT DEFAULT 0 NOT NULL
);
```

Bei der Definition der Tabelle `artikel` kann dieselbe Domäne verwendet werden, um die Fremdschlüsselspalte zu erzeugen:

```
CREATE TABLE IF NOT EXISTS artikel (
...
mwst_art mehrwertsteuerart NOT NULL REFERENCES mwst(mwst_art),
...
);
```

Interessant ist noch, dass der Domäne hier ein Standardwert („kein“) zugeordnet worden ist, der somit Bestandteil des Datentyps ist und bei der Definition der einzelnen Felder dann nicht wiederholt werden muss. Daher enthält die Fremdschlüsselfelddefinition in der Tabelle `artikel` keine weitere `DEFAULT`-Angabe.

CREATE DOMAIN

Die allgemeine Syntax lautet:

CREATE DOMAIN *domänenname*

AS *datenyp*

[DEFAULT *standardwert]*

CHECK

(VALUE *bedingung* {*[AND|OR] VALUE bedingung*}

);

8.7.2 Domänen ändern (ALTER DOMAIN)

Analog der Änderung von Tabellen gibt es auch die Möglichkeit, Domänen zu ändern. Dabei beschränken sich die Änderungen im Wesentlichen auf die Änderung des Standardwertes sowie der CHECK-Bedingung.

Änderung des Standardwertes

Der Standardwert kann durch eine Neubelegung des Wertes mit SET jederzeit beliebig geändert oder mit einem DROP komplett entfernt werden. Soll bei der Mehrwertsteuer als Standardwert der volle Mehrwertsteuersatz angenommen werden, so kann dies mit

```
ALTER DOMAIN mehrwertsteuerart
SET DEFAULT 'voll';
```

geschehen. Soll überhaupt kein Standardwert verwendet werden, so lautet die SQL-Anweisung:

```
ALTER DOMAIN mehrwertsteuerart
DROP DEFAULT;
```

Ganz analog kann die CHECK-Bedingung geändert werden. Dabei ist vor einer Änderung jeweils das Entfernen der bisherigen Bedingungen notwendig:

```
ALTER DOMAIN mehrwertsteuerart
DROP CONSTRAINT;
```

Die neue Bedingung kann dann mit einem ADD hinzugefügt werden.

```
ALTER DOMAIN mehrwertsteuerart
ADD CHECK
   (VALUE IN ('voll','teil','kein')
);
```

ALTER DOMAIN

Damit ergibt sich als Syntax für die ALTER DOMAIN-Anweisung

ALTER DOMAIN *domänenname*

[DROP DEFAULT | DROP CONSTRAINT | SET DEFAULT *ausdruck* |

ADD CHECK]

(VALUE IN *bedingung* {*[AND|OR] VALUE IN bedingung*}

);

Beachten Sie, dass die vorhandenen Werte in den existierenden Daten eines Feldes die neuen Bedingungen nicht erfüllen müssen. Schließlich wurden sie unter anderen Voraussetzungen in die Tabellen eingefügt. Je nach Datenbanksystem wird eine solche Änderung mit oder ohne Löschung vorhandener Werte übernommen.

Standardwerte können problemlos übernommen werden, da sie nur neue (INSERT) oder neu zu ändernde Datensätze (UPDATE) betreffen.

8.7.3 Domänen löschen (DROP DOMAIN)

Das Löschen von Domänen geschieht wiederum mit einer DROP-Anweisung:

```
DROP DOMAIN mehrwertsteuerart;
```

Beachten Sie aber, dass eine Domäne in Tabellen verwendet wird. Solange diese Tabellen existieren, können Sie eine Domäne nicht löschen. Das Datenbankmanagementsystem quittiert dies mit einer Fehlermeldung des SQL-Interpreters. Vielmehr müssen Sie alle Tabellen, die eine Domäne verwenden, löschen, bevor Sie die Domäne löschen.

Das Löschen der Tabelle bedeutet dabei ein wirkliches Entfernen mit einem `DROP TABLE`. Ein bloßes Löschen der Daten (`DELETE`, `TRUNCATE`) ist nicht ausreichend, da die Struktur der Tabelle, die letztlich die Domäne beinhaltet, von einem bloßen `DELETE` oder `TRUNCATE` nicht berührt wird.

DROP DOMAIN

Die Syntax zum Löschen einer Domäne lautet:

DROP DOMAIN *domänenname;*

8.7.4 Übungen

Übungen

Die Übungen hier können nur mit einem Datenbanksystem durchgeführt werden, das das Konzept der Domänen wie beschrieben unterstützt. Daher sind die Lösungen hier nur für Firebird erstellt.

1. Die Domäne `mehrwertsteuer` soll nicht durch Texte, sondern durch die Werte 1, 2 und 0 codiert werden. Der Standardwert ist 0. Erstellen Sie eine entsprechende Domäne. (Ü8.7.1)
2. Wir haben bereits verschiedene Felder verwendet, die Mengen beinhalten. Diese sollen in der Regel nicht negativ werden. Definieren Sie eine Domäne `menge`, die nur ganzzahlige, positive Werte erlaubt. (Ü8.7.2)
3. Das Geschlecht soll nur die Werte „M“ oder „W“ enthalten dürfen. Definieren Sie eine entsprechende Domäne `geschlecht`. (Ü8.7.3)
4. Löschen Sie die definierten Domänen wieder. (Ü8.7.4)

9

Unterabfragen (Sub-SELECT)

9.1 Nutzung von Unterabfragen

Der grundsätzliche Aufbau einer Abfrage mit SELECT war Thema des Kapitels 4. Danach haben wir uns mit der Änderung der Daten in der Datenbank mit den INSERT-, UPDATE- und DELETE-Anweisungen beschäftigt und schließlich in den letzten Kapiteln mit dem Aufbau der eigentlichen Datenbankstruktur.

Unterabfrage

Jetzt sollen weitere Möglichkeiten von Abfragen und Änderungen aufgezeigt werden, um komplexere Probleme zu lösen. Dafür sind sogenannte Unterabfragen in SQL vorgesehen. Eine Unterabfrage besteht aus einer eigenen SELECT-Anweisung, die wie üblich eine Menge von Datensätzen liefert. Das Besondere besteht dann darin, dass die so ermittelte Datensatzmenge unmittelbar in der eigentlichen SELECT-Anweisung weiterverwendet wird. Es werden also mithilfe einer SELECT-Anweisung ein oder mehrere (virtuelle) Datensätze erstellt, die dann zumeist als Basis für Vergleiche oder Auswahlentscheidungen in der WHERE-Klausel der übergeordneten eigentlichen SELECT-Anweisung genutzt werden (siehe Abbildung 9.1).

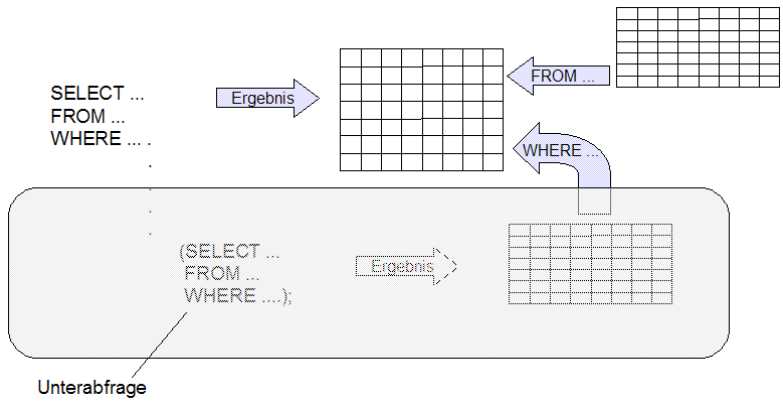


Abbildung 9.1
Das Ergebnis der Unterabfrage wird als (virtuelle) Tabelle unmittelbar verwendet.

Sub-SELECT

Da es sich bei der Unterabfrage also prinzipiell um eine eigene SELECT-Anweisung handelt, spricht man häufig auch von einem *Sub-SELECT* oder einer *Sub-Query*, ohne dass dies allerdings eigene SQL-Schlüsselwörter oder -Anweisungen sind.

Beispiel

Es soll jetzt beispielsweise die Liste aller Dozenten erstellt werden, deren Stundensatz mindestens genauso hoch ist wie der durchschnittliche Stundensatz aller Dozenten. Das Problem besteht dann darin, dass zunächst der durchschnittliche Stundensatz bekannt sein muss, um dann im zweiten Schritt zu ermitteln, ob der Stundensatz eines bestimmten Dozenten über diesem durchschnittlichen Stundensatz liegt.

1. Schritt

Bei Abfragen mit Unterabfragen sind also mindestens zwei Schritte notwendig. Im ersten Schritt wird eine Menge von Datensätzen ermittelt, die im zweiten Schritt genutzt wird. Dabei kann es sich um einen oder mehrere Datensätze handeln. Hier kann mit

```
SELECT AVG(t.Stundensatz)
FROM tbDozent t;
```

zunächst der durchschnittliche Stundensatz ermittelt werden. Das Ergebnis ist in Abbildung 9.2 dargestellt. Es handelt sich hier also um eine sehr kleine virtuelle Tabelle mit einer einzigen Spalte und einem einzelnen Datensatz.

Abbildung 9.2
Ergebnismenge der (geplanten) Unterabfrage

avg(t.Stundensatz)
14.8

2. Schritt

Das Prinzip ist jetzt immer dasselbe, egal wie klein oder groß die Ergebnismenge des ersten Schrittes ist. Im zweiten Schritt wird die eigentliche Hauptabfrage erstellt. Dabei handelt es sich zunächst um eine normale SELECT-Anweisung, die alle Datenfelder und alle Tabellen beinhaltet, die für das fertige Ergebnis benötigt werden. In unserem Beispiel sollen der Familienname, der Vorname, die Dozenten-Identifikation (DID) sowie der Stundensatz für jeden Dozenten angegeben werden,

Listing 9.1
Hauptabfrage noch ohne Unterabfrage

```
SELECT p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON (p.PID = t.PID);
```

Das Ergebnis der SELECT-Anweisung sind alle Dozenten mit ihrer DID und ihrem Stundensatz. Über den INNER JOIN wird zusätzlich aus der Tabelle tbPerson auf den Familiennamen und den Vornamen der Dozenten zugegriffen (siehe Abbildung 9.3).

Abbildung 9.3
Alle Dozenten mit ihren Stundensätzen

Familienname	Vorname	DID	Stundensatz
Weiss	Peter	812	17
Schlachter	Dieter	815	15
Weiss	Karin	821	13
Cromberg	Jörg	833	14
Bucz	Susanne	834	15

Soweit lassen sich die beiden SELECT-Anweisungen einzeln jede für sich vorbereiten. Jetzt werden sie so zusammengebaut, dass der erste Schritt als WHERE-Klausel in die zweite Anweisung eingesetzt wird. Die Ergebnisse der Unterabfrage, hier der durchschnittliche Stundensatz, werden dabei verwendet, um mit den Ergebnissen der Hauptabfrage in Beziehung gesetzt zu werden. Hier werden alle Datensätze dahin gehend überprüft, ob der Stundensatz größer ist, als der in der Unterabfrage in der ersten und einzigen Spalte stehende Wert.

Listing 9.2
Kombinierte Abfrage mit Unterabfrage

```
SELECT p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON t.PID = p.PID
WHERE t.Stundensatz >=
   (SELECT AVG(t2.Stundensatz) FROM tbDozent t2);
```

Konkret wird also für alle Datensätze geprüft, ob

```
t.Stundensatz >= AVG(t2.Stundensatz)
```

gilt. Sie sehen die Bedeutung der Alias in diesem Fall. Die Tabelle tbDozent wird zweifach verwendet. Das Alias t bezeichnet die Tabelle in der Hauptabfrage. Hier verbergen sich alle einzelnen Datensätze der Tabelle tbDozent hinter dem Alias, während mit t2 die ebenfalls auf tbDozent beruhende, aber gruppierte und mithilfe der Aggregatfunktion AVG zu einem Datensatz mit dem Durchschnittswert verdichtete virtuelle Tabelle der Unterabfrage angesprochen wird.

Es werden also nacheinander alle Stundensätze der Dozenten mit dem durchschnittlichen Stundensatz 14.8 verglichen. Dann werden als Ergebnis der WHERE-Klausel nur diejenigen Datensätze angezeigt, deren Stundensatz mindestens 14.8 beträgt. Das Ergebnis zeigt die Abbildung 9.4.

Abbildung 9.4
Ergebnis der Abfrage mit Unterabfrage

Familienname	Vorname	DID	Stundensatz
Weiss	Peter	812	17
Schlachter	Dieter	815	15
Bucz	Susanne	834	15

Eine solche Abfrage ist nicht durch eine geschickte einfache SQL-Anweisung zu ersetzen. Im Folgenden sind einige Ansätze zu sehen, die man vielleicht versuchen würde, zu verwenden. Der naheliegendste Versuch ist:

Listing 9.3
Erster nicht funktionsfähiger Versuch, eine Unterabfrage zu ersetzen

```
SELECT
   p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON t.PID = p.PID
WHERE t.Stundensatz >= AVG(t.Stundensatz);
```

Der Versuch scheitert daran, dass die Aggregatfunktion AVG() zwingend eine Gruppierung voraussetzt. Nur wenn eine Gruppierung vorhanden ist, kann mithilfe einer Aggregatfunktion eine Gruppe von Datensätzen verdichtet werden, in diesem Fall wird der Mittelwert der gruppierten Datensätze ermittelt.

Das gleiche Problem ergibt sich auch, wenn die Aggregatfunktion unmittelbar in der Datenfeldliste verwendet wird:

Listing 9.4 Zweiter nicht funktionsfähiger Versuch, eine Unterabfrage zu ersetzen

```
SELECT
   p.Familienname, p.Vorname, t.DID, t.Stundensatz,
   avg(t.Stundensatz)
FROM tbDozent t INNER JOIN tbPerson p ON t.PID = p.PID
WHERE t.Stundensatz >= AVG(t.Stundensatz);
```

Es ist also zwingend eine Gruppierung der Datensätze erforderlich, um den Mittelwert der Stundensätze ermitteln zu können. Die Gruppierung müsste allerdings derart gestaltet werden, dass alle Datensätze in die Gruppierung einbezogen werden, da der Mittelwert aller Datensätze benötigt wird. Andererseits werden aber die Einzeldatensätze ermittelt, deren Stundensatz über dem Mittelwert liegt. Die folgenden Versuche funktionieren daher zwar syntaktisch, ermitteln aber wiederum nicht alle Datensätze, sondern nur die gruppierten Datensätze.

```
SELECT
   p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON t.PID = p.PID
GROUP BY t.DID
HAVING t.Stundensatz >= AVG(t.Stundensatz);
```

oder

Listing 9.5 Erfolgreiche Gruppierung, die aber keine Einzeldatensätze mehr liefert

```
SELECT
   p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON (t.PID = p.PID)
INNER JOIN tbDozent t2 ON (t2.PID = p.PID)
GROUP BY t2.DID
HAVING t.Stundensatz >= AVG(t2.Stundensatz);
```

Eine Gruppierung über alle Datensätze wäre nur möglich, wenn es ein Datenfeld gäbe, dessen Wert in allen Datensätzen identisch ist. Das ist aber kaum die Idee einer relationalen Datenbank.

Fassen wir also zusammen: Immer wenn in mehreren Schritten zunächst eine Datenmenge ermittelt werden soll, die selbst wieder Basis für eine weitergehende Abfrage ist, ist eine Unterabfrage in Erwägung zu ziehen. Unterabfragen dürfen in der `WHERE`-Klausel zusätzlich zu bereits vorhandenen Beziehungen auftreten, sodass sich folgende Syntax ergibt:

Syntax

SELECT [DISTINCT|ALL] *ausdruck, [{, ausdruck}]*

FROM *tabelle [joinliste]*

[WHERE

(SELECT ...

)]

[GROUP BY *feldname [{, feldname}]]*

[HAVING

(SELECT ...

)]

[ORDER BY {*feldnamenliste [ASC|DESC]*}];

Sie sehen bereits an der Syntax, dass analog zu der Verwendung in der WHERE-Klausel auch eine Verwendung einer Unterabfrage in der HAVING-Klausel für gruppierte Datensätze möglich ist.

So würde in obigem Beispiel auch die folgende SQL-Anweisung zu dem Resultat in Abbildung 9.4 führen, sie wäre allerdings umständlicher und in dieser Form nicht notwendig.

Listing 9.6
Unterabfrage in der HAVING-Klausel

```
SELECT
   p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON (t.PID = p.PID)
GROUP BY t.DID
HAVING t.Stundensatz >=
   (SELECT AVG(t2.Stundensatz)
    FROM tbDozent t2);
```

Programmierung

Unterabfragen können für eine Datenbank aufwendige Operationen sein. Besteht die Möglichkeit, das Ergebnis der Unterabfrage mit einem eigenen SELECT zu ermitteln und zwischenzuspeichern, beispielsweise im Rahmen der Programmierung, kann dies einfacher und performanter sein. Unter Umständen kann das Ergebnis auch mehrfach wiederverwendet werden.

Eine Unterabfrage ist oft nur ein Weg, eine Ergebnismenge zu erzeugen und unmittelbar weiterzuverwenden. Stehen alternative Wege zur Erreichung desselben Ergebnisses zur Verfügung, sollten diese stets geprüft werden.

9.2 Unterabfragen mit Vergleichsoperatoren

Unterabfragen werden wie in unserem ersten Beispiel gern verwendet, um Vergleichswerte für die WHERE- oder HAVING-Klausel zu ermitteln. Die Logik beruht dann darauf, dass die Unterabfrage eine Zeile mit einem Datenfeld liefert. Dieses Feld stellt letztlich einen einzigen Wert dar, der als Vergleichswert verwendet werden kann und dann zumeist mit den Vergleichsoperatoren =, >, <, >= oder <= mit den Werten in anderen Datenfeldern verglichen wird.

Die Unterabfrage tritt an die Stelle eines beliebigen Ausdrucks, Datenfelds oder Literals und hat alle Eigenschaften eines Ausdrucks, wie Datentyp, Länge oder NULL-Wert. Die Grundsyntax ist:

feldname Vergleichsoperator **(SELECT ...)**

Wobei die Unterabfrage und der erste Operand auch ihre Position tauschen können. So liefert in der folgenden SQL-Anweisung die Unterabfrage die Anzahl der Kursbesuche. Da keine weitere Einschränkung erfolgt, handelt es sich um die Besuche aller Kurse:

Listing 9.7
Ermittlung aller Kurse, wenn mindestens ein Kurs besucht wird

```
SELECT KID
FROM tbKursbesuche t
WHERE
   ( SELECT COUNT(*) FROM tbKursbesuche) > 0;
```

Sinnvoller ist es, die Kurse zunächst zu gruppieren und dann die Anzahl der Teilnehmer je Kurs zu ermitteln, um dann die Kurse herauszufiltern, die tatsächlich Teilnehmer haben.

Listing 9.8
Alle Kurse mit mindestens einem Teilnehmer

```
SELECT KID
FROM tbKursbesuche t
GROUP BY t.KID
HAVING (
   SELECT COUNT(*) FROM tbKursbesuche) > 0;
```

Unterabfrage ohne Aggregation

Grundsätzlich lassen sich Unterabfragen auch nutzen, um Werte zu ermitteln, ohne dabei Aggregatfunktionen zu verwenden. Im folgenden Beispiel wird mithilfe einer Unterabfrage zunächst ermittelt, welche Kursthemen-Identifikation (`KTHID`) der Kurs „CE23“ hat (siehe Listing 9.9).

Listing 9.9
Ermittlung eines Wertes mit einer Unterabfrage

```
SELECT th.Kursthema
FROM tbKursthema th
WHERE th.KTHID =
   (SELECT t.KTHID FROM tbKurs t WHERE t.KID = 'CE23')
;
```

Dann wird der so ermittelte Wert genutzt, um ihn mit den `KTHID` in der Tabelle `tbKursthema` zu vergleichen und das eigentliche Kursthema des Kurses „CE23“ auszugeben. Sie sehen, dass hier eigentlich ein `JOIN` ausgeführt wird. Sie könnten dasselbe Ergebnis auch mit der Anweisung in Listing 9.10 erreichen.

Listing 9.10
Nutzung eines JOIN statt einer Unterabfrage

```
SELECT th.Kursthema
FROM tbKursthema th INNER JOIN tbKurs t ON (th.KTHID = t.KTHID)
WHERE (t.KID = 'CE23')
;
```

Ein wenig sieht die Nutzung der Unterabfrage hier aus wie die Nutzung eines Werkzeugs für jedes Problem, getreu dem Motto „Wenn man einen Hammer hat, sieht jedes Problem wie ein Nagel aus“. Tatsächlich ist in vielen Fällen die Performance beider Varianten gegeneinander abzuwägen und dann zu entscheiden, welche zu bevorzugen ist, auch wenn in der Regel der `JOIN` zumindest bei einem *EQUI-JOIN* zu bevorzugen ist.

Bei Unterabfragen mit Vergleichsoperatoren wird in den allermeisten Fällen ein Bezug zwischen einem Datenfeld der übergeordneten Abfrage und dem mit der Unterabfrage ermittelten Wert hergestellt. Das ist gerade der Zweck des Vergleichs.

Grundsätzlich lassen sich aber auch beide Seiten eines Vergleichs über Unterabfragen bestimmen, ohne dass ein Bezug zu einer Spalte der übergeordneten Abfrage entsteht, wie in Listing 9.11 zu sehen.

Listing 9.11
Zwei Unterabfragen in der WHERE-Klausel

```
SELECT KID
FROM tbKursbesuche t
WHERE
   (SELECT COUNT(*) FROM tbKursbesuche t2)
=
   (SELECT SUM(t3.Fehltage) FROM tbKursbesuche t3)
;
```

Es werden in diesem Fall zunächst die Kursbesuche insgesamt gezählt und danach die Fehltage aller Kursbesuche ermittelt. Sind diese identisch, erfolgt eine Ausgabe der KID, sonst ist der Kurs nicht relevant.

Weil die Ermittlung der Werte in den Unterabfragen aber in keinem Zusammenhang mit den Werten in der Gesamtabfrage steht, erfolgt auch deren Ausgabe der KID unabhängig von dem Ergebnis der Unterabfragen. Der einzige Zweck der Unterabfragen besteht hier darin, zu ermitteln, ob die beiden Anzahlen gleich sind, nicht aber darin, dies in irgendeinen Zusammenhang mit dem Endergebnis zu setzen. Der Zweck dieser Abfrage ist hier fragwürdig. Es gibt aber auch viele sinnvolle Beispiele derartiger Abfragen, die dann aber zumeist mehrere Datensätze und andere Operatoren beinhalten. Darauf soll im nächsten Abschnitt eingegangen werden.

Übungen zu Unterabfragen mit Vergleichsoperator

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Ermitteln Sie den Vornamen und Familiennamen der jüngsten Person aus tbPerson. (Ü9.2.1)
2. Ermitteln Sie die KID und den durchschnittlichen Rabattsatz aller Kurse, deren durchschnittlicher Rabattsatz mindestens so hoch ist wie der durchschnittliche Rabattsatz aller Teilnehmer aller Kurse. (Ü9.2.2)
3. Verwenden Sie jetzt die Artikeldatenbank. Ermitteln Sie die Artikelnummer (anr) und die Bezeichnung sowie den Listenpreis aller Artikel, deren Listenpreis größer als der durchschnittliche Listenpreis aller Artikel ist. (Ü9.2.3)
4. Ermitteln Sie die Artikelnummer (anr) und die Bezeichnung sowie den Listenpreis des teuersten Artikels. (Ü9.2.4)
5. Ermitteln Sie für alle Warengruppen, deren durchschnittlicher Listenpreis über dem Gesamtdurchschnitt aller Artikel liegt, die Bezeichnung der Warengruppe und deren durchschnittlichen Listenpreis. (Ü9.2.5)

9.3 Unterabfragen mit ALL und ANY

Eng verwandt mit den bisherigen Abfragen sind Vergleichsabfragen mit den beiden Operatoren ALL und ANY. ALL und ANY sind prinzipiell nur Umschreibungen für die Verwendung der Aggregationsfunktionen MIN() und MAX() zur Ermittlung des kleinsten respektive größten Wertes einer Gruppe von Werten.

Beispiel

Sollen beispielsweise alle Kursteilnehmer ermittelt werden, die bisher mindestens so viel bezahlt haben, wie derjenige Teilnehmer des Kurses „CE23“, der am meisten bezahlt hat, so kann dies mit der folgenden SQL-Anweisung ermittelt werden:

Listing 9.12
Überdurchschnittliche Beitragszahler des Kurses „CE23“

```
SELECT p.Familienname, p.Vorname, kb.GezahlterBetrag, kb.KID
FROM tbPerson p INNER JOIN tbKursbesuche kb ON (p.PID = kb.KTID)
WHERE kb.GezahlterBetrag >=
   ALL(SELECT kb2.GezahlterBetrag
      FROM tbKursbesuche kb2
      WHERE kb2.KID='CE23'
   );
```

Das Ergebnis ist in Abbildung 9.5 dargestellt.

Abbildung 9.5
Alle Teilnehmer, die mindestens so viel bezahlt haben wie das Maximum der Teilnehmer von Kurs CE23

Familienname	Vorname	GezahlterBetrag	KID
Karmann	Thomas	350	CE23
Müller	Claudia	350	CE23
Schulze	Tanja	350	CE23
Ruppert	Nicola	350	CE23
Lisewski	Bernd	350	CE24

Die Aussage der Abfrage war etwa: „Ermittle diejenigen Teilnehmer, die mindestens so viel Beitrag bezahlt haben wie ALLE (anderen, aber auch er selbst) Teilnehmer des Kurses CE23.“ Das bedeutet, dass das Maximum des von den Teilnehmern des Kurses „CE23“ bezahlten Betrags ermittelt werden muss und anschließend damit verglichen wird. Somit kann derselbe Effekt auch mit folgender SQL-Anweisung erzielt werden:

```
SELECT p.Familienname, p.Vorname, kb.GezahlterBetrag, kb.KID
FROM tbPerson p INNER JOIN tbKursbesuche kb ON p.PID = kb.KTID
WHERE kb.GezahlterBetrag >=
   (SELECT MAX(kb2.GezahlterBetrag)
      FROM tbKursbesuche kb2
      WHERE kb2.KID='CE23'
   );
```

Eng verwandt mit dem `ALL`-Operator ist der `ANY`-Operator. Mit `ANY` (übersetzt „irgendein“) werden alle Datensätze ausgewählt, deren Wert im Vergleichsfeld größer oder kleiner als irgendein Datensatz der Unterabfrage ist. Im folgenden Beispiel werden also alle Personen ausgewählt, die mehr bezahlt haben, als irgendein Teilnehmer des Kurses „CE23“ bezahlt hat. Somit wird de facto mit dem kleinsten gezahlten Betrag (also dem Minimum) verglichen und es werden alle Teilnehmer ermittelt, die NICHT den kleinsten Betrag bezahlt haben (siehe Listing 9.13).

Listing 9.13
Alle Teilnehmer, die NICHT den kleinsten Betrag bezahlt haben

```
SELECT p.Familienname, p.Vorname, kb.GezahlterBetrag, kb.KID
FROM tbPerson p INNER JOIN tbKursbesuche kb ON p.PID = kb.KTID
WHERE kb.GezahlterBetrag >
   ANY (SELECT (kb2.GezahlterBetrag)
      FROM tbKursbesuche kb2
      WHERE kb2.KID = 'CE23'
   );
```

Damit können durch Negation dieser Aussage gerade die Teilnehmer ermittelt werden, die höchstens den kleinsten Betrag aus dem Kurs „CE23“ bezahlt haben.

Listing 9.14
Alle Teilnehmer, die den kleinsten Betrag bezahlt haben

```
SELECT p.Familienname, p.Vorname, kb.GezahlterBetrag, kb.KID
FROM tbPerson p INNER JOIN tbKursbesuche kb ON p.PID = kb.KTID
WHERE NOT (kb.GezahlterBetrag >
   ANY (SELECT (kb2.GezahlterBetrag)
      FROM tbKursbesuche kb2
      WHERE kb2.KID = 'CE23'
   ));
```

Somit wird mit der folgenden SQL-Anweisung derselbe Effekt erreicht:

```
SELECT p.Familienname, p.Vorname, kb.GezahlterBetrag, kb.KID
FROM tbPerson p INNER JOIN tbKursbesuche kb ON p.PID = kb.KTID
WHERE kb.GezahlterBetrag <=
   (SELECT MIN(kb2.GezahlterBetrag)
      FROM tbKursbesuche kb2
      WHERE kb2.KID = 'CE23'
   );
```

Beachten Sie, dass die Verwendung der Funktionen `MIN` und `MAX` nicht überall möglich ist oder eine Gruppierung erfordert.

Übungen zu Unterabfragen mit ALL/ANY

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. 1.Ermitteln Sie die `KBID`, die `KID` und den gezahlten Betrag für alle diejenigen Kursbesucher, die den maximalen Betrag bezahlt haben. (Ü9.3.1)
2. 2.Schränken Sie das Ergebnis aus Ü9.3.1 so ein, dass nur noch Kursteilnehmer des Kurses „CE23“ berücksichtigt werden. (Ü9.3.2)
3. 3.Ermitteln Sie die `KID` und den durchschnittlichen von den Kursteilnehmern gezahlten Betrag von dem Kurs, bei dem dieser Durchschnitt am größten ist. (Ü9.3.3)
4. 4.Ermitteln Sie die `KTIG`, den gezahlten Betrag, den Rabatt, die Kurskennung und die Kursgebühr für den Teilnehmer, der den geringsten noch zu zahlenden Betrag (offener Betrag) hat. (Ü9.3.4)

9.4 Unterabfragen mit IN und EXISTS

Zwei weitere Operatoren arbeiten ebenfalls mit mehreren Datensätzen im Ergebnis. Mit `IN` beziehungsweise `NOT IN` kann ermittelt werden, ob der Wert eines Datenfeldes in der Hauptabfrage einem Wert eines Datenfeldes in der Unterabfrage entspricht, wenn die Unterabfrage mehrere Datensätze liefert.

Beispiel IN

Sollen beispielsweise alle Personen unserer Kursdatenbank ermittelt werden, die mindestens einen Kurs besucht haben, so muss überprüft werden, ob ihre Personen-Identifikation (`PID`) in der Liste der Kursteilnehmer-Identifikationen (`KTID`) in der Tabelle `tbKursbesuche` auftaucht. Im folgenden Beispiel wird die Ergebnismenge zusätzlich auf die Personen eingeschränkt, deren Postleitzahl mit „29“ beginnt:

Listing 9.15
Alle Kunden, die mindestens einen Kurs belegt haben

```
SELECT p.Familienname, p.Vorname, p.PLZ
FROM tbPerson p
WHERE (p.PLZ LIKE '29%') AND
      (p.PID IN (SELECT kb.KTID FROM tbKursbesuche kb));
```

Abbildung 9.6
Alle Kunden, die mindestens einen Kurs belegt haben

Familienname	Vorname	PLZ
Karmann	Thomas	29227
Klötzer	Karl	29221
Müller	Claudia	29596
Peredy	Helmut	29221
Martens	Melanie	29221
Ruppert	Nicola	29301
Sander	NULL	29223
Schulze	Tanja	29308
Magerkurth	Melissa	29336
Winter	Petra	29320

IN oder ANY

Der Operator `IN` prüft also auf inhaltliche Übereinstimmung mit mindestens einem Datensatz der Unterabfrage. Das ist nichts anderes als die Verbindung `= ANY`, die ebenfalls bedeutet, dass mindestens ein Datensatz denselben Datenwert liefern muss. Die beiden Anweisungen

```
SELECT ... FROM ... WHERE datenfeld IN (SELECT datenfeld FROM ...);
SELECT ... FROM ... WHERE datenfeld = ANY (SELECT datenfeld FROM ...);
```

sind also identisch.

NOT IN oder ALL

Umgekehrt bedeutet die Verneinung `NOT IN`, dass kein Datensatz der Unterabfrage denselben Wert haben darf, dass also alle Werte ungleich sein müssen. Daher sind auch die beiden folgenden Abfragen identisch:

```
SELECT ... FROM ... WHERE datenfeld NOT IN (SELECT datenfeld FROM
...);
SELECT ... FROM ... WHERE datenfeld != ALL (SELECT datenfeld FROM
...);
```

Streng genommen könnten Sie also auf `IN` praktisch verzichten, es ist aber recht beliebt, da es zumeist leichter zu verstehen ist als die entsprechenden Abfragen mit `ANY` oder `ALL`.

Tipp

Beachten Sie, dass jede Abfrage auf Gleichheit oder Ungleichheit sehr exakt formuliert werden muss. So dürfen bei alphanumerischen Werten keine führenden oder folgenden Leerzeichen enthalten sein (verwenden Sie `TRIM` oder eine ähnliche Funktion) und die Groß- und Kleinschreibung ist unter Umständen ebenfalls zu beachten (verwenden Sie `UPPER`, `LOWER` oder eine ähnliche Funktion).

Bei numerischen Werten sind zumeist nur ganzzahlige Werte sinnvoll, da alle anderen Angaben immer die Gefahr kleiner Rundungsdifferenzen oder unterschiedlich genauer Angaben beinhalten. Gegebenenfalls sollten Sie mit den entsprechenden Rundungsfunktionen arbeiten.

Während bei `IN` auf die Gleichheit von Werten geprüft wird, erfordert `EXISTS` nur die Existenz mindestens eines Datensatzes in der Unterabfrage, der eine bestimmte Bedingung erfüllt. Es wird also wiederum mit der Unterabfrage

eine Menge von Datensätzen ermittelt und geprüft, ob mindestens einer dieser Datensätze eine Bedingung erfüllt.

Beispiel EXISTS

Sie wollen feststellen, zu welchen Kursthemen bereits Kurse angeboten werden. Dazu ermitteln Sie eine Liste aller Kursthemen, zu denen bereits (mindestens) ein Kurs existiert.

```
SELECT kt.kursthema, kt.Kursbeschreibung
FROM tbKursthema kt
WHERE EXISTS (SELECT * FROM tbKurs k WHERE kt.KTHID = k.KTHID);
```

Listing 9.16
Kursthemen, zu denen mindestens ein Kurs existiert

Das Ergebnis ist in Abbildung 9.7 dargestellt.

kursthema	Kursbeschreibung
Einstieg mit Windows	Windows bietet die Grundlage für alle modernen und anwe...
Windows für Fortgeschrittene	Der Kurs wendet sich an PC-Benutzer, die schon etwas Erf...
Word I	Einstieg in die Textverabeitung mit Word. Grundlegende Fu...
Access I	Einstieg in das Datenbankmanagmentsystem Access. Tab...
Access II	Vertiefung der Arbeit mit MS Access. Komplexe Abfragen u...

Abbildung 9.7
Alle Kursthemen, zu denen mindestens ein Kurs existiert

Die Besonderheit des `EXISTS`-Operators ist, dass keine unmittelbare Verbindung zu einer Spalte der Hauptabfrage hergestellt wird. Die Unterabfrage steht vielmehr isoliert in der `WHERE`-Klausel. Daher ist fast immer auch in der Unterabfrage eine eigene `WHERE`-Klausel notwendig. Fehlt diese `WHERE`-Klausel in der Unterabfrage, ist die `EXISTS`-Bedingung allein durch das Vorhandensein eines einzigen Datensatzes in der angesprochenen Tabelle erfüllt.

Oft wird aber nicht nur irgendeine Bedingung in der Unterabfrage benötigt, sondern es wird Bezug auf die Datenfelder der Hauptabfrage genommen, wie bereits in Listing 9.16 und auch in Listing 9.17 gezeigt.

```
SELECT p.Familienname, p.Vorname, p.PID
FROM tbPerson p
WHERE NOT EXISTS
     (SELECT * FROM tbKursbesuche kb WHERE p.PID = kb.KTID);
```

Listing 9.17
Personen, die noch an keinem Kurs teilgenommen haben

Hier werden wiederum alle Personen ermittelt, die noch nie einen Kurs besucht haben (siehe Abbildung 9.8).

Familienname	Vorname	PID
Weiss	Peter	1
Bucz	Susanne	2
Weiss	Karin	6
Meier	Kathrin	8
Schlachter	Dieter	17
Cromberg	Jörg	26
Klever	Klaus	99

Abbildung 9.8
Alle Personen, die nie einen Kurs besucht haben

In diesem Zusammenhang spricht man auch von korrelierten Unterabfragen, worauf im nächsten Abschnitt eingegangen wird.

Übungen zu Unterabfragen mit IN/EXISTS

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Ermitteln Sie die Artikelnummer und die Artikelbezeichnung aller Artikel, für die noch keine Bestellposition vorliegt, die also noch nicht bestellt wurden. (Ü9.4.1)
2. Ermitteln Sie die Bestellnummer und das Bestelldatum aller Bestellungen, die einen Artikel der Warengruppe 9 beinhalten. (Ü9.4.2)
3. Erweitern Sie Ihre Abfrage aus Ü9.4.2 so, dass für die ermittelten Bestellungen zusätzlich die Anzahl der Bestellpositionen der Warengruppe 9 ermittelt wird. (Ü9.4.3)
4. Ermitteln Sie den Nachnamen und den Vornamen der Personen in der Kundentabelle, für die keine Bestellung existiert. (Ü9.4.4)

9.5 Synchronisierte und korrelierte Unterabfragen

Im Zusammenhang mit Unterabfragen tauchen immer wieder die Begriffe *synchronisiert* und *korreliert* (oder verbunden) auf.

Korrelierte Unterabfragen

Unter korrelierten (verbundenen) Unterabfragen sind solche Unterabfragen zu verstehen, die Feldnamen aus der Hauptabfrage in der Unterabfrage verwenden. Korrelierte Unterabfragen sind somit nicht allein ausführbar, sondern funktionieren nur im Zusammenspiel mit der jeweiligen Hauptabfrage.

Listing 9.18 Hauptabfrage und Unterabfrage sind über das Feld p.PID korreliert.

```
SELECT p.Familienname, p.Vorname, p.PID
FROM tbPerson p
WHERE NOT EXISTS
     (SELECT * FROM tbKursbesuche kb WHERE p.PID = kb.KTID);
```

Hier wird p.PID in der Unterabfrage wieder aufgegriffen. Da dieses Feld in der Hauptabfrage „definiert“ wird, ist die Unterabfrage allein nicht lauffähig. Im Gegensatz zu korrelierten Unterabfragen sind unkorrelierte (oder freie) Unterabfragen auch für sich allein ausführbar.

Synchronisierte Unterabfragen

Unter synchronisierten Unterabfragen sind solche Unterabfragen zu verstehen, bei denen in der Hauptabfrage und in der Unterabfrage dieselbe Tabelle verwendet wird. In diesem Fall ist natürlich der Einsatz von Alias zwingend erforderlich, um die Feldnamen sauber auseinanderhalten zu können. Die bereits bekannte Abfrage in Listing 9.19 stellt hier ein typisches Beispiel dar.

Listing 9.19 Synchronisierte Unterabfrage

```
SELECT p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON t.PID = p.PID
WHERE t.Stundensatz >=
     (SELECT AVG(t2.Stundensatz) FROM tbDozent t2);
```

9.6 Regeln für Unterabfragen in der WHERE-Klausel

Die bisherigen Ergebnisse für Unterabfragen sollen an dieser Stelle zusammengefasst werden. Unterabfragen werden in der `WHERE`- oder `HAVING`-Klausel einer `SELECT`-Anweisung eingesetzt, um die Menge der ermittelten Datensätze einzuschränken. Dabei gilt, dass ...

- Unterabfragen als eigene `SELECT`-Anweisung, die in Klammern gesetzt wird, formuliert werden.
- Unterabfragen mit einem Vergleichsoperator (`=`, `>`, `>=`, `<`, `<=`, `!=` oder `<>`) mit einem Datenfeld verbunden werden, wenn es sich bei dem Ergebnis der Unterabfrage um eine einzelne Zeile mit einem einzelnen Wert handelt.
- Unterabfragen mit einem Vergleichsoperator mit `ALL` oder `ANY` oder einem `IN` oder `EXISTS` eingefügt werden, wenn es sich um mehrere Zeilen handelt.
- eine Sortierung mit `ORDER BY` in einer Unterabfrage unnötig (und in den meisten Datenbanksystemen auch unzulässig) ist.
- Unterabfragen in vielen Fällen durch einen `JOIN` ersetzt werden können, aber bei Weitem nicht in allen Fällen.

9.7 Erweiterungen der Unterabfragen

Neben den bisher angesprochenen Nutzungsmöglichkeiten existieren noch eine Reihe von Erweiterungen für Unterabfragen.

Unterabfragen in der FROM-Klausel

Unterabfragen können neben der Nutzung in der `WHERE`- und der `HAVING`-Klausel auch in der `FROM`-Klausel einer `SELECT`-Anweisung genutzt werden. In diesem Fall dienen sie dazu, eine eigene virtuelle Tabelle zu schaffen, die dann ihrerseits als Datenquelle für den Rest der `SELECT`-Anweisung nutzbar ist.

Beispiel

Sie wollen beispielsweise den größten Durchschnittsbetrag aller gezahlten Beträge je Kurs bestimmen. Dafür muss zunächst in der Tabelle `tbKursbesuche` eine Gruppierung nach der `KID` vorgenommen werden, um alle Datensätze eines Kurses zusammenfassen zu können. Dann muss mit der Aggregatfunktion `AVG()` der Durchschnitt je Kurs bestimmt werden. Danach muss wiederum für diese Durchschnitte mit einer weiteren Aggregatfunktion das Maximum bestimmt werden. Das Problem liegt also im Kern in einer zweifachen Aggregation. Die naheliegende Lösung

```
SELECT MAX(AVG(GezahlterBetrag))
FROM tbKursbesuche kb
GROUP BY kb.KID;
```

funktioniert leider nicht, weil die Gruppierungsfunktion nur einstufig anwendbar ist. Mit `ANY` oder `ALL` ließe sich hier eine Lösung schaffen, wie wir bereits in einer ähnlichen Fragestellung gesehen haben.

Näherliegend und verständlicher ist aber oft eine Lösung in einer zweistufigen Vorgehensweise, bei der zunächst nach der KID gruppiert und der Mittelwert ermittelt wird. Das Ergebnis wird dann als eigene Tabelle mit einer neuen Spalte durchschnittsbetrag aufgefasst und auf diese Tabelle wird dann die Aggregatfunktion MAX() angewendet.

Listing 9.20
Die Unterabfrage in der FROM-Klausel erzeugt eine „virtuelle" Tabelle.

```
SELECT MAX(kb2.durchschnittsbetrag)
FROM (
   SELECT AVG(kb.GezahlterBetrag) AS "durchschnittsbetrag"
   FROM tbKursbesuche kb
   GROUP BY kb.KID)
AS kb2;
```

Damit wird das gewünschte Ergebnis 273,33 ermittelt. Die Alias-Zuweisung mit AS ist dabei in den meisten Datenbanken unverzichtbar, um eindeutige Benennungen zu erhalten.

In manchen Systemen, so auch in MySQL, können auch Unterabfragen verwendet werden, die nicht nur eine, sondern mehrere Spalten liefern und direkt mit vorgegebenen Werten verglichen werden. Das Prinzip entspricht dem der virtuellen oder temporären Tabellen, die in einigen Systemen auch dauerhafter erzeugt und für Abfragen innerhalb einer Session wiederverwendet werden können.

Abschließend wird noch ein Beispiel mit der neuen Funktion

```
WIDTH_BUCKET (Werte, Minimum, Maximum, Anzahl_Klassen)
```

vorgestellt. Die Funktion teilt den Wertebereich von Minimum bis Maximum in eine feste Anzahl gleich großer Klassen, also Wertebereiche, ein. Die Anzahl dieser Klassen wird über den letzten Parameter, Anzahl_Klassen, gesteuert. Dann wird für jeden Datensatz mithilfe des Eintrags im Feld Werte die Nummer der Klasse ermittelt, der der Datensatz zuzuordnen ist.

Listing 9.21
Klassifizierung der gezahlten Beträge der Kursteilnehmer (Oracle)

```
SELECT
   Klasse,
   (Klasse-1)*50 AS "ab",
   Klasse*50 AS "bis unter",
   COUNT(*) AS "Anzahl Kursbesuche"
FROM
   (SELECT WIDTH_BUCKET(GezahlterBetrag, 0, 400, 8)
   AS Klasse
   FROM tbKursbesuche)
GROUP BY Klasse
ORDER BY Klasse ASC;
```

Im vorliegenden Beispiel in Listing 9.21 werden also in der Tabelle tbKursbesuche die gezahlten Beträge untersucht. Es werden aus dem Bereich der Werte 0 bis 400 insgesamt 8 gleich große Klassen gebildet, sodass jede Klasse die Breite 50 hat. Es entsteht also eine virtuelle Tabelle, deren Anzahl an Datensätzen genau der Anzahl in der Tabelle tbKursbesuche entspricht. Diese Tabelle hat nur ein Feld mit der Klassennummer je Datensatz. Diese Tabelle wird dann in der Hauptabfrage gruppiert und für jede Gruppe wird die Anzahl der Datensätze ermittelt. Das Ergebnis ist in Abbildung 9.9 dargestellt.

KLASSE	Ab	Bis Unter	Anzahl Kursbesuche
1	0	50	2
2	50	100	3
3	100	150	1
6	250	300	5
7	300	350	2
8	350	400	5

Abbildung 9.9
Unterabfrage mit WIDTH_BUCKET

Die Funktion ist zum jetzigen Zeitpunkt zunächst in Oracle verfügbar, soll aber auch in den anderen Systemen in einer der nächsten Versionen implementiert werden.

Übungen zu erweiterten Unterabfragen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Ermitteln Sie als Vorbereitung noch ohne Unterabfrage zunächst für alle Artikel die Artikelnummer, die Artikelbezeichnung, den Listenpreis als „Nettopreis“ und einen „Bruttopreis“ mit Aufschlag des Mehrwertsteuersatzes. Runden Sie den Bruttopreis auf zwei Nachkommastellen. (Ü9.7.1)
2. Ermitteln Sie mithilfe der Tabelle aus Ü9.7.1 als Unterabfrage jetzt die `mid`, die `bnr`, die `pos` sowie die Anzahl, die Artikelnummer und den Gesamtwert jeder Bestellposition netto und brutto. (Ü9.7.2)
3. Ermitteln Sie mithilfe der Tabelle aus Ü9.7.2 als Unterabfrage für alle Bestellungen die Bestellnummer, die Kundennummer, das Bestelldatum sowie den Bestellwert netto und den Bestellwert brutto als Summe der einzelnen Positionen der Bestellung. Verwenden Sie sinnvolle Alias bei der Anzeige. (Ü9.7.3)

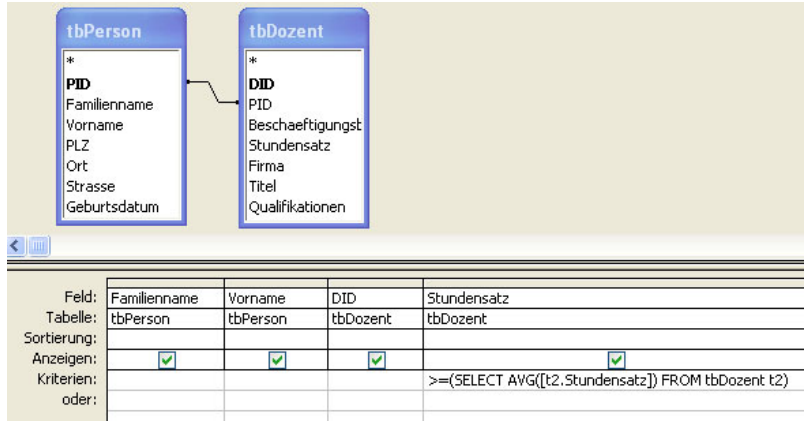
9.8 Unterabfragen mit MS Access

MS Access beherrscht Unterabfragen, allerdings sind diese nicht wirklich in die grafische Oberfläche integriert. Eine Unterabfrage wird zumeist in der WHERE-Klausel definiert. Dementsprechend ist in MS Access auch nur eine Möglichkeit vorgesehen, die komplette Unterabfrage in den Kriterien in SQL-Syntax einzugeben. So wird unser anfängliches Beispiel

```
SELECT p.Familienname, p.Vorname, t.DID, t.Stundensatz
FROM tbDozent t INNER JOIN tbPerson p ON t.PID = p.PID
WHERE t.Stundensatz >=
   (SELECT AVG(t2.Stundensatz) FROM tbDozent t2);
```

in MS Access, wie in Abbildung 9.10 dargestellt, eingegeben.

Abbildung 9.10
Abfrage mit Unterabfrage in MS Access



Das ist nicht wirklich komfortabel, aber es funktioniert wie in anderen Datenbanksystemen. Die entsprechende SQL-Anweisung, die von MS Access daraus generiert wird, zeigt bis auf die zusätzlichen Klammern eine praktisch identische SQL-Syntax:

Listing 9.22
Unterabfrage in MS Access

```
SELECT
   tbPerson.Familienname,
   tbPerson.Vorname,
   tbDozent.DID,
   tbDozent.Stundensatz
FROM tbPerson INNER JOIN tbDozent ON tbPerson.PID = tbDozent.PID
WHERE (((tbDozent.Stundensatz)>=
   (SELECT avg([t2.Stundensatz]) FROM tbDozent t2)));
```

Allerdings besteht in MS Access die Möglichkeit, bei der Erstellung einer Abfrage über die grafische Oberfläche neben den Tabellen auch auf andere Abfragen zuzugreifen. Dabei wird dann ebenfalls die SQL-Anweisung der so verwendeten Abfrage als Unterabfrage verwendet, was allerdings nur bei einer Verwendung in der `FROM`-Klausel funktioniert. Auf diese Weise können also Abfragen als Datenquellen verwendet werden, aber nicht in der `WHERE`- oder der `HAVING`-Klausel.

10

Unterabfragen in der DDL und DML

In Kapitel 9 sind Unterabfragen im Zusammenhang mit der SELECT-Anweisung besprochen worden. Unterabfragen können aber auch im Zusammenhang mit der Erstellung von Tabellen (CREATE) sowie mit der Änderung von Daten in den Tabellen (INSERT, UPDATE, DELETE) genutzt werden.

Sie werden in diesem Zusammenhang ebenfalls stets als SELECT-Anweisung formuliert, die dann als Unterabfrage beispielsweise für ein INSERT oder UPDATE die relevanten Daten bereitstellt.

10.1 CREATE mit Unterabfragen

In einigen Datenbanken wie MySQL oder Oracle besteht auch die Möglichkeit, neue Tabellen auf der Basis bestehender Tabellen mithilfe eines CREATE TABLE und einer Unterabfrage zu erstellen. Sie erinnern sich an die Erstellung einer Tabelle mit CREATE:

CREATE TABLE

(*Felddefinition* | *Integritätsbedingung* { , *[Felddefinition* | *Integritätsbedingung]*} *);*

Wird eine Unterabfrage verwendet, werden die Felder mit ihrer Definition, wie Datentyp und Länge, aus den bestehenden Strukturen gelesen und müssen bzw. dürfen nicht neu angegeben werden. Dadurch vereinfacht sich die Definition der Tabelle erheblich.

Beispiel

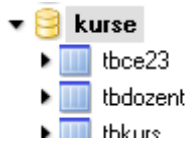
Es soll jetzt beispielsweise eine eigene Tabelle erstellt werden, in der nur die Teilnehmer des Kurses „CE23“ mit einigen wichtigen Informationen gespeichert werden. Dies kann mithilfe von MySQL mit folgendem Beispiel geschehen:

Listing 10.1
Erstellen einer neuen Tabelle für den Kurs „CE23“

```
CREATE TABLE tbce23
   (SELECT p.Familienname,p.Vorname,kb.KID,kb.Fehltage,kb.Zeugnis
   FROM tbKursbesuche kb INNER JOIN tbPerson p
      ON (p.PID = kb.KTID)
   WHERE KID = 'CE23');
```

Beachten Sie, dass Sie nach der Ausführung des Befehls gegebenenfalls in Ihrer Oberfläche eine Aktualisierung der dargestellten Tabellen durchführen müssen. Dann sollten Sie eine neue Tabelle `tbce23` sehen (siehe Abbildung 10.1).

Abbildung 10.1
Die neue Tabelle ist Teil des Schemas.



Danach können Sie den Inhalt der neuen Tabelle mit einem

```
SELECT * FROM tbce23;
```

abfragen und sollten etwa das Bild aus Abbildung 10.2 erhalten. Sie sehen, dass mit der Tabellenstruktur auch die Inhalte direkt übernommen worden sind. Diese Inhalte sind vollständige Kopien der Originaldaten und von diesen unabhängig. Damit ist jetzt in der Datenbank natürlich eine Redundanz entstanden.

Abbildung 10.2
Inhalt der neu erzeugten Tabelle

Familienname	Vorname	KID	Fehltage	Zeugnis
Karmann	Thomas	CE23	2	N
Klötzer	Karl	CE23	0	N
Martens	Melanie	CE23	5	N
Müller	Claudia	CE23	0	N
Schulze	Tanja	CE23	1	N
Winter	Petra	CE23	0	N
Ruppert	Nicola	CE23	2	N
Magerkurth	Melissa	CE23	8	N
Weiss	Peter	CE23	0	N

Schnappschuss

Eine solche Tabelle ist aber beispielsweise sinnvoll, wenn Sie einen Schnappschuss des aktuellen Standes zu einem bestimmten Stichtag benötigen, um ihn zu archivieren, an eine Außenstelle, ein Zulieferunternehmen oder andere Adressaten zu liefern. In solchen Fällen können Sie auch das aktuelle Datum leicht ergänzen, indem Sie mit `CURRENT_DATE` (oder `NOW()`) eine entsprechende Ergänzung vornehmen:

Listing 10.2
Erstellen der Tabelle mit Zeitstempel (Timestamp)

```
CREATE TABLE tbce23
   (SELECT p.Familienname,p.Vorname,kb.KID, kb.Fehltage,kb.Zeugnis,
   CURRENT_DATE()
   FROM tbKursbesuche kb INNER JOIN tbPerson p
      ON (p.PID = kb.KTID)
   WHERE KID = 'CE23');
```

Oracle

Beachten Sie, dass die Nutzung von Unterabfragen nicht in allen Datenbanken möglich ist. Außerdem weicht die Syntax von der `CREATE TABLE`-Syntax

leicht ab (siehe Kapitel 8). So würde Oracle die folgende Version der obigen CREATE TABLE-Anweisung benötigen, um dieselbe Tabelle zu erzeugen:

Listing 10.3
Erstellen einer Tabelle mit Unterabfrage in Oracle

```
CREATE TABLE tbce23 (Familienname, Vorname, KID, fehltage, Zeugnis)
   AS
   (SELECT p.Familienname,p.Vorname,kb.KID,kb.Fehltage,kb.Zeugnis
   FROM tbKursbesuche kb INNER JOIN tbPerson p
      ON (p.PID = kb.KTID)
   WHERE KID = 'CE23');
```

Damit ergibt sich eine Erweiterung der Syntax für die Erstellung von Tabellen.

CREATE TABLE

CREATE TABLE

(*Felddefinition* | *Integritätsbedingung* { , *[Felddefinition* | *Integritätsbedingung]*} **)**

| [AS] (SELECT-Anweisung);

Info

Die CREATE TABLE-Anweisung wird in dieser Form von Firebird und openBase zurzeit nicht unterstützt. In MS Access kann über eine Tabellenerstellungsabfrage ein solcher Befehl erzeugt werden. Dabei wird die SELECT-Anweisung aus der Abfrage erstellt. Wird die Abfrage dann in eine Tabellenerstellungsabfrage umgewandelt, wird intern eine entsprechende CREATE TABLE-Anweisung mit Unterabfrage erzeugt. Entsprechend steht sie in der Engine, aber wie alle CREATE TABLE-Anweisungen nicht im SQL-Fenster zur Verfügung.

Übungen zu CREATE TABLE mit Unterabfragen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Erstellen Sie eine neue Tabelle `warengruppe9`, die alle Artikel mit allen relevanten Angaben beinhaltet, die zur Warengruppe 9 gehören. (Ü10.1.1)
2. Erstellen Sie eine neue Tabelle `endkundenliste`, die eine Preisliste für die Endkunden beinhaltet. Wählen Sie dazu alle relevanten Felder aus der Tabelle `artikel` aus und berechnen Sie den Endpreis, der ausgehend von einem Nettolistenpreis die Mehrwertsteuer aufschlägt. (Ü10.1.2)
3. Erstellen Sie eine neue Tabelle `Kunden_BueroFix`, die nur die Kundennummer, den Namen und die Anschrift der Kunden der BüroFix KG (`mid=1`) beinhaltet. (Ü10.1.3)
4. Nutzen Sie jetzt die Kursdatenbank. Erzeugen Sie eine Tabelle `unbearbeiteteThemen` aus der Tabelle `tbKursthema` mit Thema, Kursbeschreibung und geplanter Dauer. (Ü10.1.4)
5. Erzeugen Sie eine Tabelle `eingesetzterDozent` aller Dozenten aus `tbDozent` von Dozenten, die mindestens einen Kurs haben. Die Tabelle soll aus der Tabelle `tbDozent` die Felder `DID`, `Beschaeftigungsbeginn` und `Stundensatz`, aus der Tabelle `tbPerson` den `Familiennamen` und `Vornamen` sowie zusätzlich aus der Tabelle `tbKurs` die Felder `Kurskennung`, `KursdauerStunden`, `Kursbeginn` und `Kursende` beinhalten. (Ü10.1.5)

10.2 UPDATE mit Unterabfragen

Die Änderung von Daten mithilfe der UPDATE-Anweisung lässt sich ebenfalls mit Unterabfragen sehr flexibel gestalten.

Beispiel

Wir wollen dazu zunächst allen Dozenten, deren Stundensatz bisher geringer als der Durchschnitt war, einen Stundensatz bezahlen, der genau dem Durchschnitt entspricht. Um dabei nicht die Originaldaten zu zerstören, legen wir zunächst eine neue Tabelle an. Dabei greifen wir wiederum auf ein CREATE TABLE mit Unterabfrage zurück.

Listing 10.4 Erstellen einer Arbeitstabelle

```
CREATE TABLE gehaltsanpassung AS
(SELECT * FROM tbDozent);
```

In MS Access (und wahlweise openBase) können Sie eine solche Tabelle stattdessen auch über die Standarddatenbankoberfläche kopieren.

Jetzt soll der durchschnittliche Stundensatz ermittelt und dann in den Datensätzen als neuer Stundensatz verwendet werden, die bisher einen niedrigeren Stundensatz aufweisen. Dazu wird in der WHERE-Klausel mit einer Unterabfrage der durchschnittliche Stundensatz ermittelt und darauf basierend die Datensätze ausgewählt, die geändert werden müssen. Dann wird in der SET-Klausel wiederum mit einer (in diesem Fall identischen) Unterabfrage der zu verwendende Stundensatz ermittelt und dem Feld Stundensatz in den ausgewählten Datensätzen zugewiesen.

Listing 10.5 Änderung des Stundensatzes auf den Durchschnittsstundensatz

```
UPDATE gehaltsanpassung g
SET g.Stundensatz = (SELECT AVG(d1.Stundensatz) FROM tbDozent d1)
WHERE g.Stundensatz < (SELECT AVG(d2.Stundensatz) FROM tbDozent d2);
```

Das Ergebnis in der Tabelle gehaltsanpassung ist in Abbildung 10.3 in Ausschnitten dargestellt.

Abbildung 10.3 Tabelle gehaltsanpassung nach dem UPDATE

DID	PID	Beschaeftigung...	Stundensatz
812	1	2003-07-01	17
815	17	2002-01-01	15
821	6	2005-09-15	14.8
833	26	2003-08-01	14.8
834	2	2001-01-01	15

Wir haben in dem Beispiel in den Unterabfragen jeweils die Tabelle tbDozent statt der Tabelle gehaltsanpassung verwendet. Das hatte nicht nur den Sinn, die Abfrage wiederholt durchführen zu können, da die Basisdaten durch die Abfrage nicht verändert werden, sondern auch den einfachen Grund, dass nur wenige Datenbanksysteme – wie beispielsweise Oracle – die Verwendung derselben Tabelle in einer Unterabfrage einer UPDATE-Anweisung erlauben. Hätte sich also die Unterabfrage in der FROM-Klausel wieder auf die Tabelle gehaltsanpassung bezogen, hätte die Abfrage in vielen Fällen schlicht nicht funktioniert, in anderen Fällen hätte sie zu immer neuen veränderten Werten geführt.

Übungen zu UPDATE TABLE mit Unterabfragen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Die Gebühren für die teureren Kurse sollen gesenkt werden. Ermitteln Sie zunächst die durchschnittliche Kursgebühr, um sich ein Bild zu machen, welche Kurse betroffen sind. (Ü10.2.1)
2. Ermitteln Sie jetzt die ID und die Gebühr aller Kurse sowie die DID des Dozenten, deren Kursgebühr über der durchschnittlichen Kursgebühr liegt. (Ü10.2.2)
3. Erhöhen Sie jetzt den Stundensatz aller Dozenten um 10 %, die in mindestens einem Kurs unterrichten, dessen Gebühr über dem Durchschnitt aller Kurse liegt. (Ü10.2.3)
4. Senken Sie jetzt die Preise aller Kurse, deren Preis über dem Durchschnitt aller Kurse liegt um 10 %. (Ü10.2.4)

10.3 INSERT mit Unterabfragen

Die INSERT-Anweisung erlaubt das Einfügen von Datensätzen in bestehende Tabellen. Wiederum können dabei neben der direkten Angabe von Werten diese Werte auch mithilfe einer Unterabfrage gewonnen und dann in die Tabelle eingefügt werden.

Direkte Werteingabe

In Kapitel 6 ist die Syntax der INSERT-Anweisung als (optionale) Liste von Feldnamen und den Feldern zugeordneten Werten beschrieben, wobei die sich entsprechende Reihenfolge der Felder und Werte entscheidend ist.

INSERT INTO *tabellenname [(feldname1, feldname2, ...)]*

VALUES (*wert1, wert2, ...);*

Die Werte werden dabei in die Felder der angegebenen Tabelle eingetragen. Werden die Feldnamen nicht angegeben, werden die Werte in der Reihenfolge der Definition in der CREATE TABLE-Anweisung eingetragen.

Beispiel

Bei Verwendung einer Unterabfrage kann diese genutzt werden, um die einzufügenden Werte bereitzustellen. Entsprechend tritt die Unterabfrage an die Stelle der VALUES-Klausel. Die Anweisung in Listing 10.6 fügt einen neuen Datensatz in die Tabelle tbKursbesuche ein. Dafür wird der Datensatz mit der PID „1“ aus der Tabelle tbPerson ausgewählt. Die entsprechende Person wird in einen neuen Kurs „CExx“ eingetragen. Die Fehltage werden auf „0“ gesetzt. Es gibt kein Zeugnis, dafür wird auch keine Gebühr verlangt.

```
INSERT INTO tbKursbesuche
   (SELECT p.PID + 1000,p.PID,'CExx',0,'N','N','Gutschein',0,0
    FROM tbPerson p
    WHERE p.PID = '1'
   );
```

Listing 10.6 Einfügen eines Kursbesuches mit einer Unterabfrage

Nach erfolgreicher Eingabe existiert ein neuer Datensatz mit der KBID „1001“. Sie können den Datensatz leicht in der Tabelle kontrollieren.

Jetzt soll das Beispiel erweitert werden und alle Personen diesem neuen Kurs „CExx“ zugewiesen werden, die bisher noch keinen Kurs besucht haben. Dazu wird zunächst der ursprüngliche Zustand wiederhergestellt und der eben eingefügte Datensatz gelöscht. Die KBID kann direkt angesprochen werden. Es reicht ein einfaches

```
DELETE FROM tbKursbesuche WHERE KBID = 1001;
```

Jetzt soll die neue INSERT-Anweisung erstellt werden. Anstatt einen Datensatz mit der festen PID=1 auszuwählen, wird wiederum eine Unterabfrage in die Unterabfrage geschachtelt, die mithilfe des EXISTS-Operators ermittelt, welche Personen noch keinen Eintrag in der Tabelle tbKursbesuche besitzen. Diese werden ermittelt und in identischer Form wie oben in die Tabelle eingetragen (siehe Listing 10.7).

Listing 10.7
Alle Personen, die noch keinen Kurs besuchen, werden für CExx eingetragen.

```
INSERT INTO tbKursbesuche
   (SELECT p.PID+1000,p.PID,'CExx',0,'N','N','Gutschein',0,0
    FROM tbPerson p
    WHERE NOT EXISTS
       (SELECT *
        FROM tbKursbesuche k2
        WHERE p.PID = k2.KTID
       )
   );
```

Das Ergebnis der Abfrage lässt sich dann in der Tabelle tbKursbesuche überprüfen. Eine SELECT-Anweisung sollte in etwa das Ergebnis in Abbildung 10.4 liefern. Sie sehen die „1000er“-PID-Datensätze, die sich jetzt alle auf den Kurs „CExx“ beziehen.

Abbildung 10.4
Tabelle tbKundenbesuche nach erfolgreichem INSERT

KBID	KTID	KID	Fehltage	Zeugnis	Selbstzahler	Zahlweise	Rabatt	GezahlterBetrag
310	25	CE17	0	J	J	Überweisung	20	260
312	4	CE24	0	N	N	Gutschein	100	250
314	4	CE17	0	J	N	Gutschein	0	280
315	5	CE17	0	J	N	Gutschein	0	280
354	34	H90	1	N	J	Überweisung	20	80
361	9	H90	0	N	N	Gutschein	0	120
412	4	CE23	2	N	N	Gutschein	0	350
413	5	CE23	0	N	J	Überweisung	100	50
415	15	CE23	5	N	J	Bar	100	250
416	10	CE23	0	N	N	Gutschein	0	350
418	31	CE23	1	N	N	Gutschein	0	350
419	32	CE23	0	N	J	Überweisung	50	300
420	24	CE23	2	N	J	Bar	0	350
421	37	CE23	8	N	J	Überweisung	150	50
423	10	CE24	0	N	J	Bar	100	0
432	15	CE24	0	N	J	Bar	100	0
438	11	CE24	0	N	N	Gutschein	0	350
442	23	CE24	0	N	J	Überweisung	50	300
999	7	CE23	0	N	J	Bar	0	50
1099	99	CExx	0	N	N	Gutschein	0	0
1026	26	CExx	0	N	N	Gutschein	0	0
1017	17	CExx	0	N	N	Gutschein	0	0
1008	8	CExx	0	N	N	Gutschein	0	0
1006	6	CExx	0	N	N	Gutschein	0	0
1002	2	CExx	0	N	N	Gutschein	0	0
1001	1	CExx	0	N	N	Gutschein	0	0

Je nach Datenbanksystem sind hier unterschiedlich komplexe Ausdrücke, beispielsweise auch Aggregatfunktionen möglich. Die prinzipielle Syntax beim Einsatz einer Unterabfrage bleibt aber gleich:

Syntax

INSERT INTO *tabellenname [(feldname1, feldname2, ...)]*

(SELECT ...);

Für weitere – auch komplexere – Beispiele sei an dieser Stelle auf das Kapitel 6 verwiesen, wo im Zusammenhang mit der `INSERT`-Anweisung bereits auf Unterabfragen in der `INSERT`-Anweisung eingegangen wurde, weil diese gerade im Zusammenhang mit dem Einfügen von Werten eine zentrale Rolle spielen.

Info

Firebird und MS Access erlauben keine Klammern um die `SELECT`-Anweisung. Lassen Sie einfach die Klammern weg, dann funktioniert die Unterabfrage wie im Standard.

Übungen zu INSERT TABLE mit Unterabfragen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine `SELECT`-Anweisung.

1. Ulrich Plate (siehe Tabelle `tbPerson`) soll als Dozent in die Tabelle `tbDozent` aufgenommen werden. Seine `DID` wird 835. Er wird zum 1.1.2009 aufgenommen. Sein Stundensatz ist 15,- €. Er ist selbstständig, Titel und Qualifikationen sind noch nicht bekannt. Nutzen Sie bei der Erstellung die Tabelle `tbPerson`, um die `PID` zu ermitteln. (Ü10.3.1)
2. Melissa Magerkurth soll ebenfalls als Dozent aufgenommen werden. Sie wird mit dem aktuellen Tag aufgenommen. Ihre `DID` wird 836. ihr Stundensatz ist der durchschnittliche Stundensatz aller anderen Dozenten. Sie ist bei der Dreher KG als Leiterin der Buchhaltung (Titel) tätig. Die Qualifikationen sind Windows und Buchhaltung. (Ü10.3.2)

10.4 DELETE mit Unterabfragen

Sie haben schon darauf gewartet? Richtig, auch die `DELETE`-Anweisung lässt sich mit einer Unterabfrage kombinieren. Die grundsätzliche Syntax ist bekannt:

DELETE

FROM *tabellenname*

[WHERE *bedingungsliste];*

Hier bietet sich für die Unterabfrage wiederum die `WHERE`-Klausel an, die dazu dient, die zu löschenden Datensätze zu identifizieren.

Beispiel

Sie wollen alle Kursteilnehmer des Kurses „CE23“ löschen, die bereits 50 % oder mehr Fehltage der Kursdauer haben. Wir verwenden dazu die Tabelle `tbce23`, die wir am Anfang dieses Kapitels mit der `CREATE TABLE`-Anweisung

erstellt haben. Dies hat den Vorteil, dass keine Daten aus den Originaltabellen gelöscht werden. Zunächst wollen wir die Unterabfrage mit einer einfachen SELECT-Anweisung testen. Dazu gehen wir bei den Kursen von täglich 8 Stunden aus. Entsprechend wird die Kursdauer in Stunden durch acht geteilt. Dies ergibt die Anzahl der Kurstage. Durch Multiplikation mit 0.5 wird dann die Hälfte der Kurstage ermittelt und mit der Anzahl der Fehltage verglichen (siehe Listing 10.8).

Listing 10.8
Alle Teilnehmer des Kurses „CE23" mit mindestens 50 % Fehlzeit

```
SELECT *
FROM tbce23
WHERE fehltage >=
   (SELECT (kursdauerStunden/8)*0.5
    FROM tbKurs k
    WHERE k.KID = 'CE23'
   );
```

Das Ergebnis sind die zwei Kursteilnehmer aus Abbildung 10.5.

Abbildung 10.5
Teilnehmer mit mindestens 50 % Fehlzeit

Familienname	Vorname	KID	Fehltage	Zeugnis
Martens	Melanie	CE23	5	N
Magerkurth	Melissa	CE23	8	N

Das sind also die beiden Teilnehmer, die jetzt mit einer Löschabfrage gelöscht werden sollen. Dazu wird einfach diese Abfrage als Unterabfrage in die WHERE-Klausel der DELETE-Anweisung eingebaut.

Listing 10.9
Löschen aller Kursteilnehmer des Kurses „CE23" mit mindestens 50 % Fehlzeit

```
DELETE FROM tbce23
WHERE fehltage >=
   (SELECT (kursdauerStunden/8)*0.5
    FROM tbKurs k
    WHERE k.KID = 'CE23'
   );
```

Nach Ausführung der Anweisung sollten die beiden Zeilen in der Tabelle tbce23 gelöscht worden sein.

Wenn Sie übrigens jetzt ein wenig aufräumen wollen und die restliche Tabelle auch noch komplett mit Struktur löschen wollen, können Sie das wie gewohnt mit einem DROP erreichen.

```
DROP TABLE tbce23;
```

Abschließend soll noch einmal die Syntax für die DELETE-Anweisung mit Unterabfrage zusammengefasst werden.

DELETE

FROM *tabellenname*

[WHERE (*SELECT-Anweisung)];*

Übungen zu DELETE TABLE mit Unterabfragen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SELECT-Anweisung.

1. Ermitteln Sie in der Tabelle `tbDozent` die ID und die höchste Kursgebühr für alle Dozenten, die nur Kurse mit Gebühren von höchstens 200,- € haben. (Ü10.4.1)
2. Löschen Sie aus der Tabelle `eingesetzterDozent` (Ü10.1.5) alle Dozenten, die nur Kurse mit Gebühren von höchstens 200,- € haben. (Ü10.4.2)
3. Löschen Sie in der Tabelle `unbearbeiteteThemen` (Ü10.1.4) alle die Kursthemen, die das Kursthema 7 zur Voraussetzung haben. (Ü10.4.3)

11

Mengenoperationen (UNION, INTERSECT, EXCEPT/MINUS)

11.1 Überblick

Sie erinnern sich, dass alle Ergebnisse von SELECT-Anweisungen Mengen sind. Das gilt für die Hauptabfragen eines einfachen SELECT genauso wie für Unterabfragen in zusammengesetzten Abfragen. Vielleicht erinnern Sie sich auch noch an die Mengenlehre in der Mathematik mit ihren typischen Mengenoperationen. Genau diese Mengenoperationen sind auch in SQL definiert.

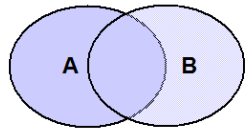
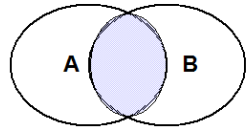
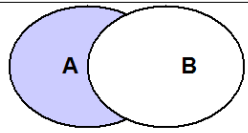
Operation	Grafik	SQL-Operator
Vereinigungsmenge $A \cup B$		UNION
Schnittmenge $A \cap B$		INTERSECT
Differenzmenge A / B		EXCEPT/MINUS

Abbildung 11.1
Mengenoperationen in SQL

Die Abbildung 11.1 zeigt die drei Grundoperationen UNION, INTERSECT und EXCEPT/MINUS, was der Bildung der Vereinigungsmenge, der Schnittmenge und der Differenzmenge entspricht. In SQL bedeutet das, dass Mengen von Datensätzen, die normalerweise mit SELECT-Abfragen gewonnen werden, anschließend miteinander vereinigt, geschnitten oder voneinander abgezogen werden.

Es geht also um die Kombination der Ergebnisse verschiedener SELECT-Anweisungen, nachdem diese ausgeführt worden sind. Diese Kombination erfolgt stets in horizontaler Richtung, also als Kombination kompletter Datensätze. Mit UNION, INTERSECT oder EXCEPT/MINUS werden also keine neuen Datensätze aus vorhandenen Datensätzen kombiniert, sondern es werden bereits vorhandene Datensätze verwendet. Stellen Sie sich einfach vor, dass jede SELECT-Anweisung eine Liste von Datensätzen erzeugt. Diese Listen werden jetzt beispielsweise untereinandergeschrieben (UNION). Die Listen können auch miteinander verglichen werden und nur die Datensätze, die in allen Listen vorhanden sind, werden verwendet (INTERSECT). Schließlich können aus einer Liste auch diejenigen Datensätze gestrichen werden, die in einer anderen Liste vorhanden sind (EXCEPT/MINUS).

Gleichartige Datensätze

Voraussetzung für eine Kombination der Datensätze in dieser Form ist jeweils, dass die Datensätze einen identischen Aufbau haben. Listen, in denen schon die Spalten unterschiedlich sind, lassen sich schwer miteinander vergleichen.

Im Normalfall werden die Datensatzmengen in den Listen jeweils durch eine SELECT-Anweisung erzeugt, sodass darauf zu achten ist, dass die SELECT-Anweisungen Datensätze desselben Formats liefern: also eine gleiche Anzahl Datenfelder mit einer gleichen Reihenfolge in Datentyp und Bedeutung, aber nicht unbedingt gleicher Herkunft. Die Daten können aus unterschiedlichen Tabellen oder sogar aus unterschiedlichen Datenbankschemata stammen.

11.2 Die Vereinigungsmenge (UNION)

UNION

Mit der SQL-Anweisung UNION werden die Ergebnismengen zweier SELECT-Anweisungen miteinander verbunden oder einfacher ausgedrückt „ohne Duplikate“ untereinandergeschrieben.

Beispiel

Der Kursbetreiber und der Büroartikelhändler wollen ihren Kundenstamm gegenüberstellen. Dabei soll zunächst eine Liste aller bekannten Personen der beiden Firmen erstellt werden. Dies bedeutet, dass die Daten aus der Tabelle tbPerson des Schemas *kurse* und aus der Tabelle kunden des Schemas *artikel* zusammen dargestellt werden. Man könnte jetzt versuchen, mithilfe eines INSERT die eine Tabelle in die andere Tabelle zu übertragen. Das kann aber zu mindestens drei Problemen führen:

- Die Daten sind strukturell unterschiedlich, da die Felder nicht gleich sind.
- Die Primärschlüssel sind doppelt.
- Die Firmen möchten überhaupt nicht, dass die Daten übertragen werden.

Daher wäre es schöner, die Daten gemeinsam darzustellen und dabei die Felder zu verwenden, die sich entsprechen. So könnte mithilfe der UNION-Anweisung eine gemeinsame „Kundenliste“ erstellt werden.

```
   SELECT p.Familienname, p.Vorname, p.Geburtsdatum, 'Kurse'
      FROM kurse.tbPerson p
UNION
   SELECT k.Nachname, k.Vorname, CURRENT_DATE, 'Artikel'
      FROM artikel.kunden k
ORDER BY 1;
```

Listing 11.1
Vereinigung der Personen aus zwei Datenbankschemata

Mit der UNION-Anweisung werden hier die Kunden und Personennamen aus zwei vollkommen unterschiedlichen Datenbankschemata zusammengeführt. Daher werden die beiden Tabellen auch mit den Schemanamen *kurse* beziehungsweise *artikel* qualifiziert. Das funktioniert nur, wenn Sie mit einem Benutzer angemeldet sind, der die Berechtigungen für beide Schemata besitzt. In MySQL ist das der Standardbenutzer für *kurse*, in Oracle *SYSTEM*. Die anderen Systeme erlauben in der hier verwendeten Oberfläche jeweils nur den Zugriff auf ein Schema, um das Beispiel nachzustellen, müssten Sie also beispielsweise jeweils die Tabelle kunden noch zusätzlich in das Schema *kurse* importieren (siehe Kapitel 3 und Anhang A).

Fehlende Spalten

Beispielhaft ist hier nicht die Spalte geburtsdatum aus der Artikeldatenbank verwendet worden, was inhaltlich sinnvoll gewesen wäre. Die Spalte ist mit CURRENT_DATE belegt worden, um zu zeigen, wie man mit einer Spalte umgehen würde, die in einer der beteiligten SELECT-Anweisungen „fehlt“, wenn man gern das entsprechende Feld aus der anderen SELECT-Anweisung hätte. Grundsätzlich besteht immer die Möglichkeit, „fehlende“ Spalten durch entsprechende Literale oder Funktionen in einer SELECT-Anweisung zu ergänzen. So lassen sich Lücken in der Struktur ausgleichen. Auch ein Casting (siehe Kapitel 5), um den Datentyp in einer SELECT-Anweisung an die andere anzupassen, kann hilfreich sein. In jedem Fall müssen alle beteiligten SELECT-Anweisungen strukturgleiche Datensätze liefern.

Die letzte Spalte ist mit den Literalen 'Artikel' beziehungsweise 'Kurse' belegt worden, um besser zu verdeutlichen, welcher Datensatz aus welchem Schema stammt.

Das Ergebnis der Anweisung ist in Abbildung 11.2 als Ausschnitt zu sehen.

Die Ergebnismenge ist hier zusätzlich sortiert worden, indem am Ende der Anweisung ein ORDER BY eingefügt wurde. Die Sortierung muss am Ende geschehen, da erst jetzt die gesamte Ergebnismenge bekannt ist.

Tipp

Sortieren Sie in einer UNION-Anweisung nach Möglichkeit mit der Spaltennummer (gezählt in der Reihenfolge der Spalten ab 1), da die Spalten in den verschiedenen SELECT-Anweisungen meistens unterschiedliche Feldnamen besitzen.

Abbildung 11.2
Ergebnis der UNION-Anweisung

FAMILIENNAME	VORNAME	GEBURTSDATUM	'KURSE'
Ammermann	Udo	16.07.08	Artikel
Bauer	Tatjana	16.07.08	Artikel
Behn	Heinrich	16.07.08	Artikel
Behrens	Cornelia	16.07.08	Artikel
Bit	Hans-Joachim	16.07.08	Artikel
Böger	Werner	16.07.08	Artikel
Bothe	Frieda	16.07.08	Artikel
Brase	Jan	16.07.08	Artikel
Bucco	Arnold	16.07.08	Artikel
Bucz	Susanne	06.04.76	Kurse
Bühren	Tamara	16.07.08	Artikel
Bühring	Uwe	16.07.08	Artikel
Büttner	Detlef	16.07.08	Artikel
Cromberg	Joerg	07.06.91	Kurse
Dreyer	Manfred	16.07.08	Artikel
Dricksen	Anja	16.07.08	Artikel
Eggers	Harald	16.07.08	Artikel
Elsner	Rudolf	16.07.08	Artikel
Falk	Hermann	16.07.08	Artikel
Förster	Ernst-Jürgen	16.07.08	Artikel
Förster	Iris	16.07.08	Artikel
Franck	Bruno	16.07.08	Artikel
Gertendorf	Rainer	16.07.08	Artikel
Grambow	Siegfried	16.07.08	Artikel
Guhr	Bernd	16.07.08	Artikel
Havenstein	Katrin	16.07.08	Artikel

UNION

Es wird Zeit, dass wir das Beispiel verallgemeinern. Die komplette Syntax für die `UNION`-Anweisung lautet:

```
SELECT (...)
UNION [ALL|DISTINCT]
SELECT(...)
```

{UNION [ALL|DISTINCT]

SELECT(...)

[ORDER BY *Spaltennummer [, Spaltennummer]]*};

Es können also mehrere `UNION`-Anweisungen kombiniert werden. Mit der zusätzlichen Angabe `ALL` oder `DISTINCT` kann entschieden werden, ob doppelte Datensätze angezeigt werden sollen oder nicht. Dazu folgt noch einmal ein Beispiel, das außerdem eine Sortierung beinhaltet. Petra Winter ist sowohl Kundin in unserer Artikeldatenbank als auch Kursteilnehmerin und somit in der Tabelle `tbPerson` enthalten. Daher liefert

```
SELECT p.Familienname, p.Vorname, p.Geburtsdatum, 'Kurse'
  FROM kurse.tbPerson p
  WHERE p.Familienname LIKE 'W%'
UNION ALL
SELECT k.Nachname, k.Vorname, CURRENT_DATE, 'Artikel'
  FROM artikel.kunden k
  WHERE k.nachname LIKE 'W%'
ORDER BY 1;
```

Listing 11.2
Alle Personen aus beiden Datenbanken, die mit „W“ beginnen

zwei Datensätze für Petra Winter. Die Einschränkung auf alle Personen, die mit W beginnen, ist übrigens nur der Übersichtlichkeit wegen erfolgt, bereits das Listing 11.1 erzeugt die doppelten Datensätze. Die Abbildung 11.3 zeigt das Ergebnis der neuen `UNION`-Anweisung.

FAMILIENNAME	VORNAME	GEBURTSDATUM	'KURSE'
Weiss	Peter	07.11.63	Kurse
Weiss	Karin	05.10.62	Kurse
Weiss	Peter	02.03.74	Kurse
Wieckenberg	Peter	16.07.08	Artikel
Winter	Petra	30.12.89	Kurse
Winter	Petra	16.07.08	Artikel

6 rows returned in 0,00 seconds CSV Export

Abbildung 11.3
Vereinigungsmenge mit Duplikaten

Jetzt sind die beiden Datensätze für „Petra Winter“ aus Sicht des SQL-Interpreters noch keine doppelten Datensätze, da nur der Name und der Vorname gleich sind, die beiden anderen Spalten aber unterschiedliche Werte beinhalten. Reduzieren Sie die Abfrage auf die beiden Namensfelder, sehen Sie den Unterschied.

```
   SELECT p.Familienname, p.Vorname
      FROM kurse.tbPerson p
      WHERE p.Familienname LIKE 'W%'
UNION ALL
   SELECT k.Nachname, k.Vorname
      FROM artikel.kunden k
      WHERE k.nachname LIKE 'W%'
ORDER BY 1;
```

Listing 11.3
UNION mit doppelten Datensätzen

Listing 11.4
UNION ohne doppelte Datensätze

```
   SELECT p.Familienname, p.Vorname
      FROM kurse.tbPerson p
      WHERE p.Familienname LIKE 'W%'
UNION DISTINCT
   SELECT k.Nachname, k.Vorname
      FROM artikel.kunden k
      WHERE k.nachname LIKE 'W%'
ORDER BY 1;
```

Die Ergebnisse der beiden Abfragen sind in Abbildung 11.4 und Abbildung 11.5 dargestellt. Sie sehen dabei auch, dass sich das DISTINCT auch auf Duplikate aus einem einzelnen SELECT bezieht, denn während das Duplikat „Petra Winter“ aus der UNION der beiden Schemata entstanden ist, war „Peter Weiss“ schon innerhalb der Kursdatenbank ein „Duplikat“.

Abbildung 11.4
Vereinigungsmenge mit Duplikaten (UNION ALL)

FAMILIENNAME	VORNAME
Weiss	Peter
Weiss	Karin
Weiss	Peter
Wieckenberg	Peter
Winter	Petra
Winter	Petra

6 rows returned in 0,00 seconds CSV Export

Abbildung 11.5
Vereinigungsmenge ohne Duplikate (UNION DISTINCT)

FAMILIENNAME	VORNAME
Weiss	Karin
Weiss	Peter
Wieckenberg	Peter
Winter	Petra

4 rows returned in 0,00 seconds CSV Export

Standard: DISTINCT

Wird weder ALL noch DISTINCT angegeben, gilt in den meisten Systemen der Standardwert DISTINCT. In Oracle wie in MS Access darf das Schlüsselwort DISTINCT überhaupt nicht verwendet werden, der UNION-Operator ist ohne zusätzliche Angabe von ALL automatisch „DISTINCT“. UNION ist in den meisten Datenbanksystemen realisiert, im Gegensatz zu den jetzt zu besprechenden Operatoren INTERSECT und EXCEPT/MINUS, die von den hier besprochenen Systemen in der Oberfläche zurzeit überhaupt nur Oracle bereitstellt.

11.3 Die Schnittmenge (INTERSECT)

Ähnlich wie bei der Erstellung der Vereinigungsmenge zweier SELECT-Anweisungen kann mit der Anweisung INTERSECT die Schnittmenge zweier SELECT-Anweisungen ermittelt werden.

Beispiel

Sollen die gemeinsamen Kunden unseres Büroartikelversands und des Kursanbieters ermittelt werden, so kann dies in Oracle mit der folgenden SQL-Anweisung geschehen:

```
   SELECT p.Familienname, p.Vorname, p.geburtsdatum
      FROM kurse.tbPerson p
INTERSECT
   SELECT k.nachname, k.Vorname, k.geburtsdatum
      FROM artikel.kunden k
ORDER BY 1;
```

Listing 11.5
Gemeinsame Kunden in beiden Schemata

Unsere Tests mit den beiden UNION-Anweisungen haben bereits gezeigt, dass nur Petra Winter in beiden Datenbeständen enthalten ist. Daher ist es nicht überraschend, wenn wir jetzt das Ergebnis in Abbildung 11.6 erhalten.

Familienname	Vorname	geburtsdatum
Winter	Petra	1989-12-30

Abbildung 11.6
Schnittmenge der beiden Kundentabellen

Die Syntax der INTERSECT-Anweisung ist derjenigen der UNION-Anweisung sehr ähnlich. Das Schlüsselwort INTERSECT wird zwischen zwei SELECT-Anweisungen gesetzt. Dieses Konstrukt kann beliebig oft wiederholt werden.

SELECT (...)

INTERSECT [ALL]

SELECT(...)

{INTERSECT [ALL]

SELECT(...)

[ORDER BY *Spaltennummer [, Spaltennummer]]*};

Das Schlüsselwort INTERSECT wird nicht von allen Datenbanksystemen unterstützt. Oracle (und das hier nicht besprochene DB2) kennen aber INTERSECT. Werden INTERSECT- und UNION-Anweisungen kombiniert, werden zunächst die INTERSECT-Anweisungen ausgeführt.

Info

Es gibt auch die `INTERSECT ALL`-Anweisung, die Duplikate in der Ergebnismenge zulässt.

Beispiel:

```
WEISS
WINTER
WINTER
INTERSECT ALL
WINTER
WINTER
```

liefert

```
WINTER
WINTER
```

im Gegensatz zu einem einfachen `WINTER` ohne `ALL`.

11.4 Die Differenzmenge (MINUS/EXCEPT)

Die dritte Mengenoperation ist die Bildung der Differenzmenge. Die Differenzmenge ist nicht symmetrisch. Es werden vielmehr die Datensätze der ersten Menge (erste `SELECT`-Anweisung) ermittelt und von diesen werden die Datensätze der zweiten Menge (zweite `SELECT`-Anweisung) „abgezogen".

Beispiel

Sie wollen nur die Kursteilnehmer ermitteln, die nicht Kunden des Büroartikelversands sind:

Listing 11.6 Personen aus der Kursdatenbank, die keine Kunden der BüroFix sind

```
   SELECT p.Familienname, p.Vorname, p.geburtsdatum
      FROM kurse.tbPerson p
MINUS
   SELECT k.nachname, k.vorname, k.geburtsdatum
      FROM artikel.kunden k;
```

Dies ergibt folgerichtig die Datensätze in Abbildung 11.7.

Die Syntax (für Oracle) lautet:

SELECT (...)

MINUS

SELECT(...)

{MINUS

SELECT(...)]

[ORDER BY *Spaltennummer [, Spaltennummer]*}};

Familienname	Vorname	geburtsdatum
Weiss	Peter	1963-11-07
Bucz	Susanne	1976-04-06
Karmann	Thomas	1954-08-04
Klötzer	Karl	1971-03-13
Weiss	Karin	1962-10-05
Weiss	Peter	1974-03-02
Meier	Kathrin	1981-05-03
Schmidt	Karl	1949-06-25
Müller	Claudia	NULL
Lisewski	Bernd	1960-06-06
Peredy	Helmut	1956-02-23
Schlachter	Dieter	1961-02-02
Martens	Melanie	1911-02-17
Ruppert	Nicola	1962-02-25
Sander	NULL	1953-02-05
Cromberg	Jörg	1991-06-07
Schulze	Tanja	1992-11-09
Magerkurth	Melissa	1951-09-04
Plate	Ulrich	1986-12-02
Klever	Klaus	1981-04-02

Abbildung 11.7
Alle Kursteilnehmer, die nicht Kunden der BüroFix sind

Info

SQL sieht im Original das Schlüsselwort `EXCEPT` statt `MINUS` vor. `EXCEPT` wird beispielsweise von DB2 unterstützt. Dort gibt es auch ein `EXCEPT ALL`, das im Sinne einer echten Subtraktion funktioniert. Liefert die erste Abfrage beispielsweise zwei identische Datensätze und die zweite Abfrage einen weiteren identischen Datensatz, würde bei einem `EXCEPT ALL` ein Datensatz übrig bleiben.

```
WINTER
WINTER
EXCEPT ALL
WINTER
```

ergibt somit

```
WINTER
```

11.5 Besonderheiten der Datenbanksysteme

MySQL

MySQL bietet nur den Operator `UNION` mit den Zusätzen `DISTINCT` und `ALL` an. Als Standard wird `DISTINCT` verwendet. `INTERSECT` wie auch `EXCEPT/MINUS` sind in der aktuellen Version noch nicht verfügbar. Durch Qualifizierung mit dem Schemanamen, also `kurse.tbPerson.PID` statt nur `tbPerson.PID`, sind Auswertungen über verschiedene Schemata möglich.

MS Access

In MS Access steht ebenfalls nur der Operator `UNION` beziehungsweise `UNION ALL` zur Verfügung. Die Eingabe muss direkt in das SQL-Fenster der Abfrage erfolgen. Die grafische Benutzeroberfläche bietet hier keine besondere Unterstützung an. MS Access kennt immer nur ein Schema, sodass eine Auswertung über Schemata hinweg nicht möglich ist.

Oracle

Oracle kennt alle drei Operatoren `UNION`, `INTERSECT` und `MINUS` (also nicht `EXCEPT`). `UNION` entspricht `UNION DISTINCT`, wobei es das Schlüsselwort `DISTINCT` nicht gibt. Zusätzlich steht `UNION ALL` zur Verfügung. Wenn Sie in Oracle über mehrere Schemata hinweg auswerten wollen, müssen Sie die entsprechenden Rechte haben. Hier können Sie sich als *SYSTEM* anmelden, müssen dann aber immer mit dem Schemanamen qualifizieren, also `kurse.tbPerson.PID` statt nur `tbPerson.PID`.

Firebird

In Firebird steht ebenfalls nur der Operator `UNION` beziehungsweise `UNION ALL` zur Verfügung. Die Eingabe muss direkt in das SQL-Fenster der Abfrage erfolgen. Firebird kennt zwar verschiedene Schemata, die hier verwendete Oberfläche lässt aber keine Auswertung über Schemata hinweg zu.

openBase

openBase kennt alle drei Operatoren, also `UNION`, `INTERSECT` und `EXCEPT`. Problematisch ist nur deren Verwendung im Abfragefenster. Eigentlich handelt es sich bei den mit `UNION` oder einem der anderen Operatoren zusammengesetzten `SELECT`-Anweisungen ebenfalls um Abfragen, wie sie im Abfragefenster abgearbeitet werden können. Diese Anweisungen liefern schließlich auch ein Ergebnis in Form einer Menge von Datensätzen. Das sieht openBase aber keineswegs genauso, die Abfragen im Abfragefenster funktionieren in der Regel nicht. Bei Problemen gibt es eine andere Möglichkeit. Geben Sie die SQL-Anweisung im Fenster EXTRAS/SQL ein. Hier werden alle drei Operatoren ausgeführt. Das Problem ist jetzt allerdings, dass in diesem Fenster keine Ergebnisse angezeigt werden. Fügen Sie daher ein `CREATE VIEW` an, also beispielsweise wie in Listing 11.7.

Listing 11.7
Erzeugung eines VIEW mit einem UNION-Operator

```
CREATE VIEW "keineDozenten" AS
      (SELECT p."PID", p.Geburtsdatum", 'Person' FROM "tbPerson" p
   UNION
      SELECT d."PID", CURRENT_DATE, 'Dozent' FROM "tbDozent" d);
```

Aktualisieren Sie danach in der Ansicht die Tabellen. Das Ergebnis können Sie sich dann wiederum im Abfragefenster mit einem einfachen

```
SELECT * FROM "keineDozenten";
```

ansehen.

openBase kennt immer nur ein Schema, sodass eine Abfrage über Schemata hinweg so nicht möglich ist.

11.6 Zusammenfassung

Mit den Mengenoperationen können Ergebnisse verschiedener `SELECT`-Anweisungen miteinander in Beziehung gesetzt werden. Sie können in beliebiger Menge miteinander kombiniert werden. Die Unterstützung ist in den meisten Datenbanksystemen eher spärlich. Dies entspricht auch der Bedeutung in der Praxis, die eher gering ist.

Die Reihenfolge der Abarbeitung der `UNION`-, `INTERSECT`- und `MINUS`-Anweisungen richtet sich im Normalfall nach der angegebenen Reihenfolge von links nach rechts beziehungsweise von oben nach unten.

Es gibt allerdings Optimierungen, die versuchen, zunächst kleinere Mengen zu ermitteln und zu verarbeiten. Daher hat dann `INTERSECT` Vorrang vor den beiden anderen Anweisungen. Auch wird oft ein `UNION DISTINCT` mit Vorrang vor einem `UNION ALL` behandelt.

Die Reihenfolge der Abarbeitung lässt sich außerdem natürlich durch Klammern beeinflussen.

11.7 Übungen

Übungen

Erstellen Sie für die folgenden Aufgaben jeweils eine SQL-Anweisung.

1. Ermitteln Sie die Personen-ID und das Geburtsdatum aller Personen aus der Kursdatenbank und verbinden Sie diese mit den Dozenten mit deren Personen-ID. Fügen Sie statt des Geburtsdatums das aktuelle Datum ein. Markieren Sie mit einer zusätzlichen Spalte die Herkunft der Datensätze als „Person“ beziehungsweise „Dozent“. Es sollen nur unterschiedliche Datensätze angezeigt werden. (Ü11.6.1)
2. Ermitteln Sie alle Postleitzahlen und Orte, aus denen die Personen der Kursdatenbank oder die Kunden der Artikeldatenbank stammen. Fügen Sie zunächst als zusätzliche Spalte eine Markierung hinzu, aus der Sie erkennen können, ob ein Datensatz aus der Kursdatenbank oder der Artikeldatenbank stammt. Sortieren Sie Ihr Ergebnis nach Postleitzahlen und innerhalb der Postleitzahlen nach der Herkunftsdatenbank. Lassen Sie Duplikate zu. (Ü11.6.2)
3. Entfernen Sie aus der Abfrage in Ü11.6.2 die Markierung der Herkunftsdatenbank und erstellen Sie eine Abfrage ohne Duplikate. (Ü11.6.3)
4. Ermitteln Sie die Personen-ID aus den Tabellen `tbPerson` und `tbDozent`, die in beiden Tabellen enthalten sind. (Ü11.6.4)
5. Ermitteln Sie die Personen-ID aus den Tabellen `tbPerson` und `tbDozent`, die nur in der Tabelle `tbPerson` enthalten sind. (Ü11.6.5)

12

Benutzer, Rechte und Zugriffsschutz

Mit diesem Kapitel verlassen wird den engeren Bereich der für die Datenbearbeitung und Schemabearbeitung definierten SQL-Befehle. Wir haben bisher noch eine Reihe von Befehlen ausgelassen, die weniger mit der Definition des Schemas und mit der unmittelbaren Abfrage und Änderung von Daten in einer relationalen Datenbank zu tun haben und zumeist als Data Control Language, also SQL-DCL, bezeichnet werden.

Trotzdem sind sie für das Funktionieren und die Performance einer Datenbank unter Umständen entscheidend. Dazu gehören sowohl die Verwaltung von Rechten (`GRANT`, `REVOKE`), die Steuerung der Transaktionen (`COMMIT`, `ROLLBACK`) wie auch eine Reihe im engeren SQL-Standard nicht definierter Anweisungen. Obwohl diese im engeren SQL-Standard teilweise nicht einmal erwähnt sind, sind sie gleichwohl De-facto-Standards zur Steuerung der physischen Datenverwaltung und Optimierung des Datenzugriffs geworden.

Diese Steuerung der physischen Datenhaltung gehört streng genommen nicht zum Aufgabenumfang von SQL, da hier im Sinn der 3-Ebenen-Architektur nicht die von SQL adressierte logische Ebene, sondern die physische Ebene betroffen ist.

Die Praxis hat allerdings gezeigt, dass gerade eine Optimierung des Zugriffs auf häufig verwendete Informationen zu den wesentlichen Anforderungen gehört. Zunächst soll es in diesem Kapitel aber um die Berechtigungen gehen.

12.1 Schutz der Informationen

Stellen Sie sich vor, jeder Benutzer könnte auf alle Daten einer Datenbank frei zugreifen. Das würde zwar zunächst viele Probleme, die aus Passwörtern und sonstigen Zugriffsschutzmechanismen resultieren, vermindern, wäre aber das sichere Ende jeder umfangreicheren Datenbank.

Die Bedenken kommen dabei aus zwei Richtungen:

- Schutz sensibler Informationen vor Verbreitung und Missbrauch und
- Schutz wertvoller Informationen vor unabsichtlicher Zerstörung oder Verfälschung.

Datenschutz

Datenbanken enthalten Informationen, die zumeist nicht von beliebigen Nutzern der Datenbank in vollem Umfang genutzt werden dürfen. Denken Sie in einem Unternehmen nur an sensible Daten wie Gehälter, Umsätze oder Rabatte. Denken Sie an eine Internetdatenbank, auf die im Netz freier Zugriff durch beliebige Nutzer möglich wäre. Die Datenbank wäre dem Angriff in jeder Form ausgesetzt. Dies ist nicht nur in Unternehmen mehr als unerwünscht.

Fehlbedienung

Bedrohungen für eine Datenbank müssen aber nicht immer durch mutwillige Angriffe, Spionage oder andere gezielte Aktionen böswilliger Nutzer entstehen. Probleme können auch einfach durch unsachgemäße Eingaben, Änderungen oder Löschungen von Benutzern entstehen. So kann ein unerfahrener Mitarbeiter in gutem Glauben Datensätze von Bestellungen löschen, die er für irrelevant hält, weil ihm Informationen fehlen. Er kann Kundendaten erfassen, die bereits vorhanden sind und so ungewollt Duplikate erzeugen. Er kann aber auch durch Bedienfehler Rabatte verändern oder eine Lieferadresse falsch bestimmen. Alles dies geschieht schlicht aus Unwissenheit um geschäftliche Regeln und Prozesse.

Privilegien

Der Zugang zur Datenbank muss daher genau auf die Bedürfnisse, Rechte und Fähigkeiten der Benutzer abgestimmt werden. Den Benutzern müssen Rechte eingeräumt werden, damit sie ihre Arbeit sinnvoll erledigen können und ihnen müssen Rechte genommen werden, um Vertraulichkeit und Qualität der Daten zu schützen. Diese Rechte oder Berechtigungen werden manchmal auch in der Übersetzung des englischen Wortes „privileges“ als Privilegien bezeichnet. Privileg klingt so herrschaftlich und machtherrlich. Das Einräumen von Privilegien hat aber nichts mit Vergabe von Macht, sondern ausschließlich mit der Abstimmung der Zugriffsrechte auf Informationen mit den betrieblichen, privaten oder sonstigen Notwendigkeiten zu tun.

Die technische Organisation der Vergabe von Rechten beginnt stets mit dem Datenbankadministrator. Bei Einrichtung des Datenbankmanagementsystems wird ein erster Benutzer eingerichtet, der Superuser, der alle Rechte erhält. Ausgehend von diesem Superuser werden dann weitere Benutzer und Benutzergruppen eingerichtet und ihnen werden die benötigten Rechte zugewiesen.

12.2 Benutzer und Benutzergruppen

Damit die Anwender von der Datenbank unterschieden werden können, müssen zunächst Benutzer eingerichtet werden. Unter Benutzern sind dabei „Konten“ zu verstehen, die dann verschiedene Rechte erhalten.

Beispiel

Soll beispielsweise ein Vertriebsmitarbeiter Meier künftig die Personen der Kursdatenbank pflegen dürfen, so muss er sich identifizieren können. Dafür wird zunächst der Benutzer `meier` selbst erzeugt.

```
CREATE USER meier;
```

Allgemein wird ein Benutzer also mit einer `CREATE USER`-Anweisung erzeugt. Dieser Identifikation wird im Normalfall eine Authentifikation mit einem Passwort hinzugefügt.

CREATE USER

CREATE USER *benutzername [IDENTIFIED BY [PASSWORD] passwort]*

{ , *benutzername [IDENTIFIED BY [PASSWORD] passwort]*};

Um eine bessere Einbettung der Datenbank in ein gesamtes Anwendungssystem zu erlauben, bieten die meisten Systeme darüber hinaus eine externe Authentifikation beispielsweise über einen Betriebssystemmechanismus oder ein anderes System an.

Rollen

Mit SQL3 ist das Konzept der `ROLE`, also der Benutzerrolle oder Benutzergruppe eingeführt beziehungsweise erheblich erweitert worden. Die Grundidee ist dabei, die aus Unternehmen bekannte Trennung zwischen einem konkreten Mitarbeiter und der Stelle, die er im Unternehmen besetzt, auch auf die Informationen zu übertragen, die dieser Stelle zur Verfügung gestellt werden sollen. Die Stelle beschreibt alle Tätigkeiten, Rechte und Pflichten innerhalb der Organisation. Sie ist im Sinne der Organisation die eigentliche Basis für die Aufbau- und Ablauforganisation. Sie kann auch die Basis für Stellenausschreibungen bilden. Der konkrete Mitarbeiter oder die konkrete Mitarbeiterin besetzt dann die Stelle und füllt sie mit allen Rechten und Pflichten aus.

Benutzergruppen

In SQL übernimmt die Rolle die Funktion der Stelle mit allen Rechten und Pflichten. Dementsprechend werden die Rechte an die Rolle gebunden, die somit aus Sicht der Datenbank den eigentlichen „Benutzer“ darstellt. Dann werden den konkreten Mitarbeitern Identifikations-IDs zugeordnet, mit denen sie sich anmelden können, und diese IDs den Rollen zugeordnet. Somit können einer Rolle mehrere Mitarbeiter zugeordnet werden. Damit kann einer Rolle ein Mitarbeiter aber beispielsweise auch sein Stellvertreter zugewiesen werden. Auch für größere Organisationseinheiten wie Gruppen, Abteilungen oder Projektteams kann eine Rolle definiert werden, der dann die IDs der Mitglieder zugeordnet werden. Dies erklärt auch die Bezeichnung einer Rolle als Benutzergruppe. Ergänzend zu diesem Konzept können auch teilweise schon Rollen- oder Gruppenhierarchien mit entsprechender Vererbung der Rechte definiert werden.

Aber Vorsicht, diese Konzepte werden typischerweise in Datenbanksystemen benötigt, die von größeren Organisationen genutzt werden. In kleineren Systemen werden sie nicht mit größter Priorität behandelt. Im Gegenteil, oft werden kostenlose oder günstige „Sparversionen“ auch größerer Systeme gerade ohne diese Konzepte angeboten, um bei breiterem Einsatz dann entsprechende Lizenzen für die kompletten Datenbanksysteme anbieten zu kön-

nen. Entsprechend sind diese Konzepte auch in den in diesem Buch als Beispiele verwendeten Systemen nicht allzu umfangreich – und recht unterschiedlich – implementiert.

12.3 Benutzer entfernen (DROP USER)

DROP USER

Entsprechend dem Vorgehen bei Schemata, Tabellen und anderen Datenbankobjekten können auch Benutzer mit einer DROP-Anweisung entfernt werden.

DROP USER *benutzername [CASCADE];*

RENAME USER

Der Zusatz CASCADE führt dazu, dass auch alle Informationen, die mit dem Benutzer verbunden sind, in der Datenbank gelöscht werden. Dies ist bei Benutzern, die zu Testzwecken eingerichtet werden, sinnvoll, sollte sonst aber mit großer Vorsicht gehandhabt werden.

Info

Teilweise bieten Datenbanken auch die Möglichkeit, Benutzer umzubenennen, was den Vorteil hat, dass die Rechte erhalten bleiben. MySQL bietet hier beispielsweise:

RENAME USER *alternname* ***TO*** *neuername****;***

12.4 Rollen einrichten

CREATE ROLE

Die Idee des Rollenkonzeptes ist es, dass der Benutzer ausschließlich zur Identifikation und Authentifikation eines Benutzers genutzt werden soll. Ist der Benutzer so akzeptiert, werden dann aufgrund der dem Benutzer zugeordneten Rolle seine Berechtigungen bestimmt. Um eine Rolle zu definieren, genügt zumeist ein

CREATE ROLE *rollenname;*

Rolle – Benutzer

Die Zuordnung der Benutzer zu den Rollen geschieht dann mit einer speziellen Version der GRANT-Anweisung:

GRANT *rollenname TO benutzername* { *, benutzername*}*;*

Womit die angegebenen Benutzer der Rolle zugeordnet werden. Damit kann ein Benutzer durch mehrfache Angabe der Anweisung auch mehreren Rollen zugeordnet werden. Die Rechte eines Benutzers ergeben sich dann aus der Summe der einzelnen Rechte der Rollen.

12.5 Rollen löschen

DROP ROLE

Rollen können wie die anderen Datenbankobjekte mit einer DROP-Anweisung gelöscht werden:

DROP ROLE *rollenname;*

Bei der Löschung einer Rolle werden alle damit verbundenen Berechtigungen ebenfalls gelöscht. Die Benutzer, die den Rollen zugeordnet sind, bleiben erhalten.

12.6 Rechte einrichten (GRANT)

GRANT

Benutzer einzurichten und zu entfernen, ist die eine Sache. Die andere Sache ist, ihnen auch tatsächlich Rechte zuzuordnen. Der zentrale Befehl zur Einrichtung von Berechtigungen ist die `GRANT`-Anweisung. Die Grundidee ist: „**Was** darf, **womit** durch **wen** gemacht werden und **unter welchen Bedingungen**."

In die Welt von SQL übersetzt bedeutet das:

GRANT *rechte*

ON *datenbankobjekte*

TO *benutzerliste*

WITH *optionen;*

Beispiel

Soll beispielsweise dem Benutzer „meier" das Recht eingeräumt werden, Daten aus der Tabelle `tbPerson` abzufragen, einzufügen und zu ändern, so kann dies mit einer `GRANT`-Anweisung wie in Listing 12.1 dargestellt geschehen.

```
GRANT SELECT, INSERT, UPDATE
ON TABLE kurse.tbPerson
TO meier
WITH GRANT OPTION;
```

Listing 12.1
Vergabe von Rechten an den Benutzer „meier"

Rechte für Tabellen

Bei der Ausgestaltung der Rechte für Tabellen als Datenbankobjekte, also im Wesentlichen der SQL-Anweisungen, gibt es erwartungsgemäß erhebliche Unterschiede. Die wesentlichen Rechte sind aber bereits im ursprünglichen Standard definiert und nur minimal verändert worden:

- `SELECT [(Datenfeldliste)]`
- `INSERT [(Datenfeldliste)]`
- `DELETE`
- `UPDATE [(Datenfeldliste)]`
- `REFERENCES [(Datenfeldliste)]`

Oder einfach

- `ALL`

Diese Rechte sind sinnvoll für Tabellen und `VIEWS` als Datenbankobjekte zu vergeben. `SELECT`, `INSERT`, `DELETE` und `UPDATE` sind selbsterklärend und beziehen sich auf die entsprechenden SQL-Anweisungen. `REFERENCES` bezieht sich auf das Recht, eine Tabelle anzulegen, die ein Datenfeld der Tabelle als Fremdschlüssel verwendet. Bei einigen Rechten können Sie zusätzlich die Spalten in einer Datenfeldliste angeben, auf die sich das Recht beziehen soll. Beachten Sie auch den Einfluss von Änderungen der Datenbankstruktur auf die Rechte. So kann es Ihnen beispielsweise bei Oracle passieren, dass Sie zwar ein `UPDATE`-Recht auf eine Tabelle haben, das sich aber nach der Ergän-

zung um eine Spalte nicht auf die neue Spalte der Tabelle bezieht. Generell sollten Sie die Rechte nach Datenbankstrukturänderungen (`CREATE`, `ALTER`, `DROP`) erneut vergeben, was angesichts des Aufwandes bei größeren Systemen für ein automatisiertes Vorgehen spricht.

Andere Datenbankobjekte

Für andere Datenbankobjekte als Tabellen sind auch andere Rechte sinnvoller. So erlaubt das Recht `EXECUTE` das Ausführen von Prozeduren und Funktionen, `TRIGGER` erlaubt den Triggereinsatz und `USAGE` erlaubt die Aktivierung von Zeichensätzen. Daneben können spezielle Rechte aber letztlich für fast alle Datenbankobjekte gewährt werden. Die `TO`-Liste gibt die Liste der Rollen (und oft alternativ der Benutzer) an, denen die Rechte gewährt werden sollen.

GRANT OPTION

Der Zusatz `WITH` erlaubt die Angabe weiterer Optionen. Die bekannteste ist die `GRANT OPTION`. Der Zusatz `WITH GRANT OPTION` bewirkt, dass ein Benutzer die übertragenen Rechte selbst wieder an andere Datenbankbenutzer weitergeben darf.

Andere Optionen können sich auf die Verwendung von Ressourcen beziehen und beispielsweise die Anzahl der Anmeldungen, der Befehle pro Stunde, der Änderungsbefehle pro Stunde oder der Ergebniszeilen pro `SELECT` betreffen.

> **Rechte für Rechtevergabe**
>
> Die Vergabe von Rechten erfordert oft bestimmte eigene Rechte. So können in Oracle beispielsweise nur Rechte vom Benutzer `SYSTEM` sowie von Benutzern vergeben werden, deren Rolle entweder `ADMIN` oder `DBA` ist oder die in der `RESOURCE`-Rolle eine `GRANT OPTION` auf die eigenen Objekte haben.

12.7 Rechte entziehen (REVOKE)

REVOKE

Rechte müssen natürlich auch wieder entzogen werden können. Der zentrale Befehl zum Entzug von Berechtigungen ist die `REVOKE`-Anweisung. Die Grundidee und die Syntax entsprechen genau der `GRANT`-Anweisung. Die Frage lautet: „**Was** darf **womit** durch **wen** nicht mehr gemacht werden."

In die Welt von SQL übersetzt bedeutet das dann:

REVOKE *rechte*

ON *datenbankobjekte*

FROM *benutzerliste;*

Bei der Ausgestaltung der verschiedenen Rechte und Datenbankobjekte richten sich die verschiedenen Hersteller normalerweise auch nach ihrer jeweiligen Implementierung der `GRANT`-Anweisung.

Beispiel

Es soll testweise ein neuer Benutzer `meier` erzeugt werden, der die Tabelle `tbPerson` vollständig bearbeiten darf, während er bei den Dozenten neben der Auswertung nur einfügen und ändern darf. Anschließend werden dem Benutzer alle Rechte entzogen und der Benutzer wird wieder entfernt (siehe Listing 12.2).

```
CREATE USER meier IDENTIFIED BY hase;
GRANT SELECT, INSERT, UPDATE ON kurse.tbDozent TO meier;
GRANT ALL ON kurse.tbPerson TO meier;
REVOKE ALL ON kurse.tbDozent FROM meier;
REVOKE ALL ON kurse.tbPerson FROM meier;
DROP USER meier;
```

Listing 12.2
Erzeugen eines Benutzers und Löschen mit Rechtevergabe

Zunächst wird der Benutzer mit einer Authentifikation über ein Passwort erzeugt. Dann werden ihm Abfrage-, Einfüge- und Änderungsrechte an der Tabelle `tbDozent` gewährt. Anschließend erhält er alle Rechte an der Tabelle `tbPerson`. Diese Rechte werden ihm dann jeweils wieder genommen und schließlich wird der Benutzer komplett entfernt.

12.8 Benutzerkonzepte verschiedener Datenbanken

12.8.1 Benutzerkonzept in MySQL

Mit der Einrichtung einer MySQL-Datenbank wird ein Superuser angelegt, der als `root` bezeichnet wird. Es muss zwar kein Passwort vergeben werden, das sollte aber natürlich im Normalfall immer erfolgen, da die Datenbank sonst für alle Anwender total offen ist. Alle weiteren Benutzer werden dann auf Basis dieses ersten Benutzers eingerichtet, der alle dazu notwendigen Rechte hat. Sie können den Befehl

```
CREATE USER meier;
```

unmittelbar in die gewohnte MS Query Oberfläche eingeben, um einen neuen Benutzer `meier` zu erzeugen. Sie können die Eingabe in MySQL auch über die Kommandooberfläche durchführen. Wichtig ist nur, dass Sie die entsprechenden Rechte besitzen, die Sie als Benutzer `root` immer haben sollten. Die Kommandooberfläche können Sie unmittelbar oder über das Administrationswerkzeug MySQL Administrator starten. Sie finden den Befehlszeilen-Client über die Menüleiste (siehe Abbildung 12.1).

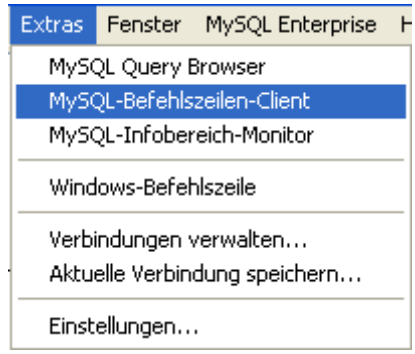
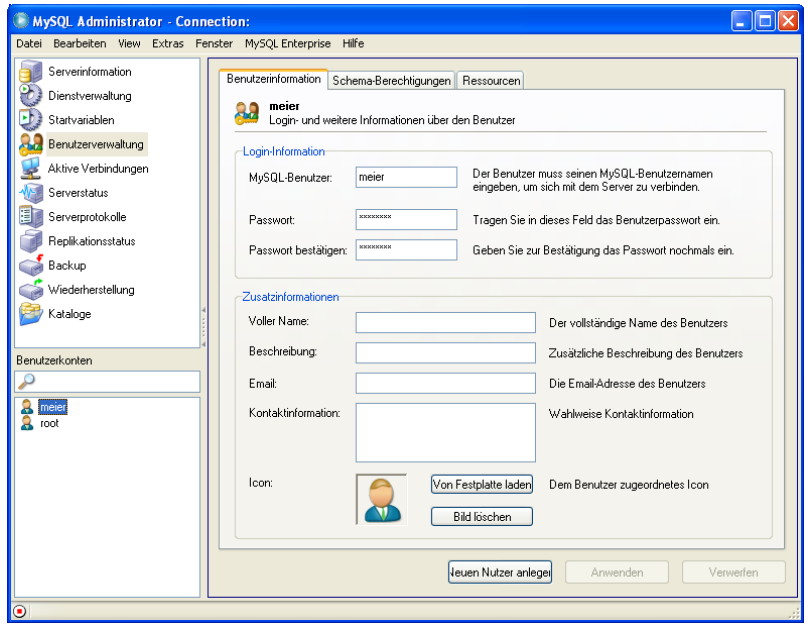


Abbildung 12.1
Auswahl innerhalb von MySQL Administrator

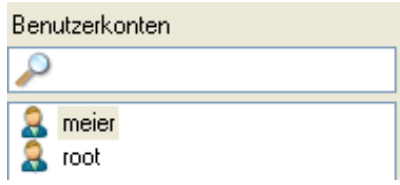
Nach Eingabe der CREATE USER-Anweisung (denken Sie an das Semikolon und an eine anschließende Exit-Anweisung) können Sie dann die Einrichtung des Benutzers in der Administrationsoberfläche prüfen. Sie können den Benutzer natürlich auch über die Oberfläche interaktiv einrichten, aber wir wollten es direkt mit SQL tun. Die Abbildung 12.2 zeigt die Administrationsoberfläche von MySQL im Überblick. Oben links können Sie die Benutzerverwaltung auswählen.

Abbildung 12.2
Überblick über MySQL Administrator



Sie können die Einrichtung des neuen Benutzers leicht prüfen, indem Sie die verfügbaren Benutzer (User Accounts) zunächst im Fenster unten links aktualisieren (REFRESH).

Abbildung 12.3
Benutzer in der Administrationsoberfläche



Der neue Benutzer hat damit aber noch keine Rechte, irgendetwas innerhalb der Datenbank zu tun, außer sich anzumelden. Wenn Sie den Benutzer auswählen, sehen Sie bei den Berechtigungen zunächst noch keine Einträge (Abbildung 12.4). Sie können jetzt für jedes Schema, beispielsweise unsere *Artikel*-Datenbank, die verfügbaren Rechte anzeigen. Hier finden Sie viele bereits bekannte SQL-Anweisungen wieder, deren Verwendung für das Schema *Artikel* Sie jetzt dem neuen Benutzer `meier` erlauben können.

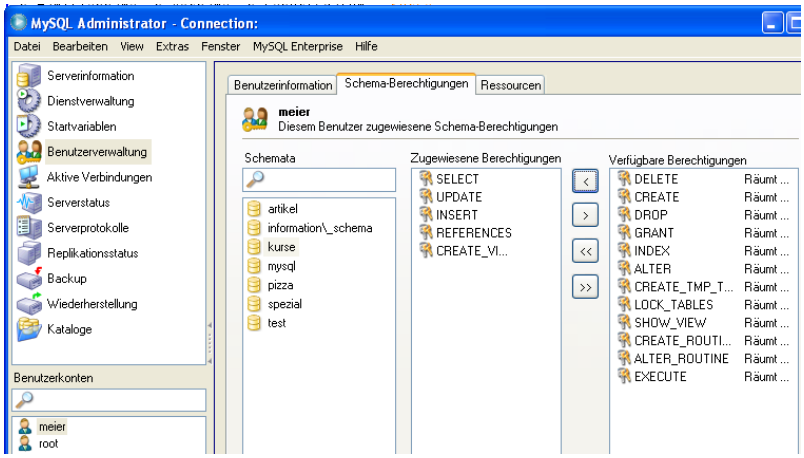


Abbildung 12.4
Rechteverwaltung eines Benutzers (privileges)

Sie sehen hier die klassischen Rechte `SELECT`, `UPDATE`, `DELETE`, `INSERT`, aber auch weitere Rechte, die sich oft auf bestimmte Datenbankobjekte beziehen. Ein echtes Rollenkonzept im oben angesprochenen Sinn kennt diese MySQL-Version nicht.

In MySQL wird das Passwort eines Benutzers mit einer eigenen `SET PASSWORD`-Anweisung geändert. Mit

SET PASSWORD

SET PASSWORD = PASSWORD('*neues Passwort*');

wird das Passwort des aktuellen Benutzers geändert. Für andere Benutzer kann das Passwort mit

SET PASSWORD FOR *benutzer* = *PASSWORD('neues Passwort');*

geändert werden, sofern der aktuelle Benutzer die Berechtigung dazu hat.

12.8.2 Benutzerkonzept in Oracle

Oracle verwendet aufgrund seines oft unternehmensweiten Einsatzes in der großen Version ein sehr differenziertes Konzept. Mit Einrichtung eines Oracle-Datenbanksystems werden zwei Benutzer als Superuser eingerichtet, *SYSTEM* und *SYS*. Die Passwörter sind bei Einrichtung des Systems bekannt und sollten schnell geändert werden. Beiden Benutzern wird die Benutzerklasse DBA (Database Administrator) zugeordnet, die die größtmöglichen Rechte gewährt.

Rollen

Oracle hat ein Rollenkonzept für die Benutzer, das jedem Benutzer den Status

- `CONNECT`,
- `RESOURCE`,
- `DBA` oder
- `ADMIN`

zuordnet.

Diese Rollen beinhalten standardmäßig eine große Menge von Rechten und stellen letztlich eine komfortable Möglichkeit dar, einem neu erzeugten Benutzer eine sinnvolle Menge an Rechten direkt zuzuordnen.

Listing 12.3 Einrichten eines Benutzers in Oracle

```
CREATE USER meier IDENTIFIED BY meinpasswort;
GRANT ADMIN TO meier;
```

Mit den beiden SQL-Anweisungen aus Listing 12.3 weisen Sie dem neu erzeugten Benutzer alle Rechte zu, die man normalerweise als Administrator benötigt. Teilweise historisch bedingt ist die kombinierte Möglichkeit, eine derartige Rollenzuweisung zusammen mit der Erzeugung des Benutzers in einer GRANT-Anweisung bei der Erzeugung des Benutzers vorzunehmen.

```
GRANT [RESOURCE|DBA|ADMIN]
TO benutzer [IDENTIFIED BY passwort];
```

CONNECT

Der Status CONNECT ist für den typischen Nutzer der Datenbank vorgesehen. Grundsätzlich dürfen die entsprechenden Daten (sofern die Rechte für die entsprechenden Tabellen gegeben sind) abgefragt und geändert werden:

- Daten abfragen (SELECT)
- Daten ändern (INSERT, UPDATE, DELETE)
- VIEWs erzeugen

RESOURCE

Die Rolle RESOURCE ist für Datenbankentwickler vorgesehen. Sie dürfen grundsätzlich alle Operationen durchführen, die mit CONNECT erlaubt sind, und zusätzlich innerhalb eines Schemas (der Domäne):

- CREATE TABLE, CREATE INDEX, CREATE CONSTRAINT, sowie die entsprechenden Anweisungen mit ALTER und DROP
- GRANT und REVOKE, um die Rechte an den Objekten des Schemas anderen Benutzern geben oder entziehen zu können

Damit können komplette Schemata wie unsere Kursdatenbank oder die Artikeldatenbank jeweils von einem Benutzer oder einer Gruppe verwaltet werden.

DBA

Was noch fehlt, ist die Möglichkeit, Benutzer mit der Rolle DBA zu verwalten, die dann auch ein

- CREATE SCHEMA, DROP SCHEMA;
- CREATE USER, ALTER USER und DROP USER;
- GRANT und REVOKE sowie
- Datensicherungs- und Überwachungsoperationen und
- alle Befehle der RESOURCE-Rolle für alle Schemata

durchführen dürfen, also die Datenbankverwaltung außerhalb eines einzelnen Schemas.

ADMIN

Die letzte Rolle ist noch ADMIN, die letztlich die Befehle umfasst, die ein DBA hat ohne die Kommandos für die RESOURCE-Rolle. Diese Rolle ist für rein (technische) Administratoren gedacht, die sich nicht mit der Verwaltung des Datenbankschemas befassen. Hier liegt letztlich auch der Unterschied zwischen den Superusern SYS und SYSTEM.

Oracle hat das Konzept der Rollen, wie es jetzt im Standard definiert worden ist, bereits umgesetzt. Hier erhält jeder Benutzer nach seiner Anmeldung alle Rechte, die ihm wie bisher als einzelnem Benutzer zugewiesen wurden. Zusätzlich werden ihm alle Rechte zugewiesen, die in seinen sogenannten benutzerdefinierten Rollen festgelegt sind. Um diese benutzerdefinierten Rollen festzulegen, wurde ein

CREATE ROLE *rollenname [IDENTIFIED BY passwort | NOT IDENTIFIED];*

eingeführt. Die entsprechenden Anweisungen zur Änderung des Passwortes und zum Löschen einer Rolle lauten dann:

ALTER ROLE *rollenname [IDENTIFIED BY passwort | NOT IDENTIFIED];*

DROP ROLE *rollenname;*

Mit der bekannten `GRANT`-Anweisung können dann die Rechte flexibel auf die Rollen verteilt werden. Wir können das komplexe System hier nicht im Detail betrachten, aber versuchen, dessen Nutzen an einem Beispiel zu verdeutlichen.

Beispiel

Die drei Mitarbeiter `meier`, `stolze` und `wegmann` sind alle Mitarbeiter im Vertrieb. Alle Vertriebsmitarbeiter sollen in der Tabelle `kunden` Abfragen durchführen und Datensätze einfügen und ändern dürfen. Bestellungen dürfen alle drei Mitarbeiter beliebig ändern. Der Gruppenleiter `wegmann` darf auch Kunden löschen sowie alle anderen Aktionen auf dieser Tabelle durchführen. Die Artikel dürfen von allen Vertriebsmitarbeitern gelesen werden. Die Mitarbeiter `mueller` und `neumayer` verwalten `artikel` und dürfen damit alle Operationen durchführen. Ihr gemeinsamer Chef `ahrens` soll alles dürfen, was seine Mitarbeiter dürfen. Nachdem alles eingerichtet ist, findet ein Mitarbeiterwechsel statt. Jetzt soll `stolze` keinen Zugriff mehr auf diesen Teil der Datenbank haben und durch `altmann` ersetzt werden.

Um die Mitarbeiter und die Rollen anzulegen, sollten Sie über die entsprechenden Rechte verfügen. Daher sollen Sie sicherheitshalber zunächst den Benutzer `SYSTEM` verwenden.

Listing 12.4
Beispiel für Rollen- und Benutzerverwaltung mit Oracle

```
CREATE USER meier IDENTIFIED BY pwmeier;
CREATE USER stolze IDENTIFIED BY pwstolze;
CREATE USER wegmann IDENTIFIED BY pwwegmann;
CREATE USER mueller IDENTIFIED BY pwmueller;
CREATE USER neumayer IDENTIFIED BY pwneumayer;
CREATE USER ahrens IDENTIFIED BY pwahrens;
CREATE ROLE vertrieb;
CREATE ROLE vertriebsleiter;
CREATE ROLE artikelverwaltung;
CREATE ROLE chef;
```

Jetzt müssen Sie Zugriff auf die Datenbankobjekte, also die Tabellen, haben. Dafür bietet sich der Benutzer `artikel` an.

Listing 12.5 Beispiel für Rechteverwaltung mit Oracle

```
GRANT SELECT, INSERT, UPDATE ON kunden TO vertrieb;
GRANT ALL ON bestellung, bestell_position TO vertrieb;
GRANT SELECT ON artikel TO vertrieb;
GRANT ALL ON kunden TO vertriebsleitung;
GRANT ALL ON artikel TO artikelverwaltung;
GRANT vertrieb TO meier, stolze, wegmann, chef;
GRANT vertriebsleitung TO wegmann, chef;
GRANT artikelverwaltung TO mueller, neumayer, chef;
GRANT chef TO ahrens;
REVOKE vertrieb FROM stolze;
GRANT vertrieb TO altmann;
```

Sie sehen in Listing 12.5 wie zunächst die Benutzer und dann die Rollen erzeugt werden. Die Benutzer entsprechen den tatsächlichen Mitarbeitern, die Rollen geben die Position im Betrieb wieder. Danach werden die Rechte an den Tabellen den Rollen zugeordnet. Der Vertrieb erhält seine Zugriffsrechte. Der Vertriebsleiter erhält seine zusätzlichen Rechte. Dann erhält die Rolle `artikelverwaltung` ihre Rechte. Jetzt müssen die Rollenrechte noch den eigentlichen Mitarbeitern zugeordnet werden. Dazu dienen die nächsten drei `GRANT`-Anweisungen. Beachten Sie, dass die Rechte dabei nicht nur den einzelnen Benutzern, sondern auch der übergeordneten Rolle `chef` zugewiesen werden. Dadurch sammeln sich dort alle Rechte. Schließlich wird die Stelle des Chefs besetzt, indem die Rechte an der Rolle dem Benutzer `ahrens` zugeordnet werden. Dies ist die eigentliche „Stellenbesetzung" im Sinn der Datenbank.

Natürlich hätte man im Fall des Vertriebsleiters und des Chefs auf eine eigene Rolle verzichten können, da es sich nur um einen Benutzer handelt. Das Rollenkonzept ist aber flexibler, weil hier beispielsweise durch

```
GRANT vertriebsleiter TO meier;
```

der Vertriebsleiter einen Stellvertreter `meier` mit denselben Rechten erhält. Auch im Falle eines Wechsels ist das Rollenkonzept sehr flexibel, wie die Ersetzung von `stolze` durch `altmann` beweist.

Das Passwort eines Benutzers kann in Oracle mit der `ALTER USER`-Anweisung geändert werden. Die Syntax entspricht der `CREATE USER`-Anweisung:

ALTER USER

ALTER USER *benutzername IDENTIFIED BY PASSWORD passwort;*

12.8.3 Benutzerkonzept in Firebird

Firebird wurde zunächst als Datenbank für Programme entwickelt. Daher hat man sich hier für eine eigene Sicherheitsdatenbank entschieden, die als *security2.fdb* im Firebird-Verzeichnis liegt. Sie enthält nur die beiden Tabellen `host_info` und `users`, ist aber über die Oberfläche nicht zugänglich.

GSEC

Der transparenteste, wenn auch nicht der einfachste Zugangsweg ist ein Tool, das zusammen mit anderen im Firebird-Verzeichnis *bin* gespeichert wird. Das ist bei Ihnen normalerweise *C:\Programme\Firebird\Firebird_2_0\bin*. Sie können auf Kommandozeilenebene darauf zugreifen. Geben Sie ein:

```
gsec -user SYSDBA -password masterkey
```

Das Tool meldet sich mit:

```
GSEC>
```

Sie können jetzt einen neuen Benutzer `meier` mit dem Passwort `hase` einfügen. Zusätzlich können Sie noch seinen Vornamen „Horst“ und den Nachnamen „Meier“ angeben:

```
GSEC> add meier -pw hase -fname Horst -lname Meier
```

Ihre Eingaben können Sie mit

```
GSEC> display
```

überprüfen.

Um das Löschen eines Benutzers zu demonstrieren, wird noch ein zweiter Benutzer angelegt und wieder gelöscht.

```
GSEC> add meier2 -pw hase -fname Horst -lname Meier
GSEC> delete meier2
```

Sie verlassen das Tool wieder mit:

```
GSEC> quit
```

Das Tool arbeitet nur auf der Sicherheitsdatenbank. Diese dient der Identifikation und Authentifikation der Benutzer. Firebird verwendet diese Datenbank sozusagen als Eingangskontrolle.

Info

Sie können die Benutzer auch über die Oberfläche mit SERVER/USER SECURITY pflegen, was aber wenig übersichtlich ist, oder aber einfach über die Oberfläche mit dem Symbol USERS im linken Fenster ganz unten.

Ist der Benutzer einmal identifiziert, können die mit SQL-Anweisungen bearbeiteten Rollen für die Zuordnung der Berechtigungen verwendet werden.

So kann für die Vertriebsgruppe eine Rolle `vertrieb` angelegt werden und der Rolle können Rechte zugewiesen werden. Weiter können mit `GRANT` auch die Benutzer der Rolle zugeordnet werden (siehe Listing 12.6).

Listing 12.6
Beispiel für den Umgang mit Rollen in Firebird

```
CREATE ROLE vertrieb;
GRANT ALL ON tbPerson TO vertrieb;
GRANT vertrieb TO meier;
REVOKE SELECT ON tbKurs FROM vertrieb;
DROP ROLE vertrieb;
```

Die Rollen lassen sich auch über die Datenbankobjekte anzeigen (siehe Abbildung 12.5).

Wie bereits im Beispiel zu sehen ist, werden auch in Firebird die Rechte mit der `REVOKE`-Anweisung entfernt. Die Rollen werden mit `DROP ROLE` gelöscht. Benutzer müssen getrennt über die beschriebenen Wege gelöscht werden.

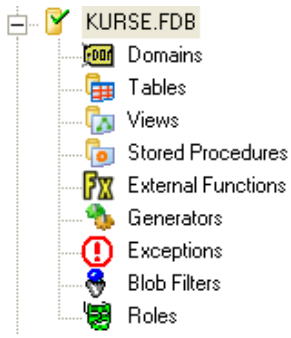


Abbildung 12.5
Datenbankobjekte einschließlich der Rollen

12.8.4 Benutzerkonzept in MS Access

Das Sicherheitskonzept, das den meisten anderen Systemen am nächsten kommt, heißt in MS Access „Sicherheit auf Benutzerebene".

MS Access verwendet für die Verwaltung der Benutzerberechtigungen eine eigene Datei, die normalerweise die Endung *.mdw* besitzt. Bei der Installation wird eine Standarddatei *System.mdw* angelegt. Diese Datei beinhaltet alle Benutzer und Benutzergruppen. Sie können der Verwaltung in der Standarddatei beitreten oder eine komplett neue Berechtigungsdatei aufbauen. Wählen Sie dazu EXTRAS/SICHERHEIT/ARBEITSGRUPPENADMINISTRATOR (erst ab Access 2003). Erstellen Sie eine neue Datei und beachten Sie die Sicherheitshinweise. Sie haben jetzt eine neue Systemdatei, in der alle Berechtigungen aufgebaut werden können.

Wählen Sie SICHERHEIT/BENUTZER- UND GRUPPENKONTEN, um den Inhalt zu bearbeiten. Im Standard richtet MS Access die beiden Benutzergruppen *Administratoren* und *Benutzer* ein. Der einzige Benutzer, der Superuser, heißt in MS Access *Administrator* und ist Mitglied beider Gruppen.

Um eine echte Anmeldung zu erzwingen, müssen Sie dem Benutzer *Administrator* ein Passwort zuordnen. Wählen Sie dazu ADMINISTRATIONSKENNWORT ÄNDERN (früher: Anmeldungskennwort ändern). Das bisherige Passwort ist leer, Sie können ein neues Passwort eingeben und dieses bestätigen. Verlassen Sie danach MS Access komplett. Nach dem erneuten Aufruf und der Anmeldung bei Ihrer Datenbank wird das Passwort abgefragt.

Sie können diesen Mechanismus wieder abstellen, indem Sie wiederum SICHERHEIT/BENUTZER- UND GRUPPENKONTEN wählen, dann BENUTZER und dann für den Benutzer *Administrator* die Option KENNWORT LÖSCHEN auswählen.

MS Access kennt sowohl Benutzer als auch Benutzergruppen, wie bereits aus der Oberfläche deutlich wird. Beiden können wiederum über die Oberfläche Berechtigungen zugewiesen werden, sodass die Gruppen mit den Rollen teilweise vergleichbar sind. Die Anweisungen lassen sich allerdings nur über die Engine, also in programmierter Form, nicht über die Oberfläche verwenden.

12.8.5 Benutzerkonzept in openBase

openBase verfügt über ein dem Standard sehr nahes Benutzerkonzept mit Benutzern, Rollen und Berechtigungen. Alle Befehle können im Fenster unter EXTRAS/SQL eingegeben werden. Bei der Einrichtung des Systems wird als Superuser der Benutzer *sa* ohne Passwort eingerichtet.

openBase bietet in seinem SQL-Fenster die Möglichkeit, weitere Benutzer direkt einzurichten. Mit der Angabe

```
CREATE USER meier PASSWORD hase [ADMIN];
```

lässt sich ein Benutzer `meier` einrichten. Die Administratorrechte können bei der Erstellung des Benutzers direkt mit der Option `ADMIN` eingerichtet werden, was hier aber gar nicht notwendig ist. Für einen einmal erzeugten Benutzer kann das Passwort jederzeit geändert werden:

```
ALTER USER meier SET PASSWORD "loeffel";
```

Damit ist ein Benutzer erzeugt. Jetzt können auch Rollen in ähnlicher Weise generiert werden. Dazu genügt ein:

```
CREATE ROLE vertrieb;
```

Damit ist die Rolle `vertrieb` erzeugt. Die Zuordnung von Benutzern zu Rollen erfolgt wie in Firebird mit einer besonderen Form der `GRANT`-Anweisung:

```
GRANT vertrieb TO meier;
```

Diese SQL-Anweisung führt dazu, dass dem Benutzer `meier` die Rolle `vertrieb` zugeordnet wird. Dann können der Rolle die benötigten Privilegien zugeordnet werden, beispielsweise das Recht, ein `SELECT` auf der Tabelle `tbPerson` durchzuführen:

```
GRANT SELECT ON "tbPerson" TO vertrieb;
```

Damit müsste auch der Benutzer `meier` eine `SELECT`-Anweisung auf der Tabelle `tbPerson` ausführen können. Der Test ist einfach. Im laufenden Betrieb kann mit

```
CONNECT USER meier PASSWORD "loeffel";
```

die Anmeldung mit einem neuen Benutzer erfolgen. Die Nutzung der Anführungsstriche für das Passwort ist wegen der Problematik mit der Groß-/Kleinschreibung ratsam.

Testen Sie es. Melden Sie sich an und führen Sie im Abfragefenster den `SELECT` aus. Versuchen Sie auch ein `SELECT` auf einer anderen Tabelle. Es muss mit einer entsprechenden Fehlermeldung zurückgewiesen werden. Wenn Sie übrigens danach zu Ihrem ursprünglichen Benutzer zurückkehren wollen, können Sie das durch ein `CONNECT` mit dem Superuser *sa* erreichen:

```
CONNECT USER sa PASSWORD "";
```

Jetzt können Sie dem `vertrieb` seine Rechte wieder entziehen:

```
REVOKE SELECT ON "tbPerson" FROM vertrieb;
```

Damit hat die Rolle `vertrieb` keine Rechte mehr und kann auch entfernt werden:

```
DROP ROLE vertrieb;
```

Schließlich kann auf demselben Weg auch der `USER` wieder gelöscht werden:

```
DROP USER meier;
```

12.9 VIEW als Zugriffsschutz

Wir haben bereits über die Definition von Benutzersichten in Form des `VIEW` gesprochen. Wie der Name Benutzersicht schon andeutet, handelt es sich dabei eigentlich um ein ideales Konzept, gerade den lesenden aber auch den ändernden Zugriff (mit den diskutierten Einschränkungen) festzulegen.

Bereits durch die Definition des `VIEW` wird die Datensicht eingeschränkt. Die Wahl der Spalten wie auch der Datensätze über eine entsprechende `SELECT`-Klausel erlaubt eine sehr flexible Festlegung der für einen Benutzer oder eine Benutzergruppe zur Verfügung zu stellenden Informationen. Da ein `VIEW` ebenfalls ein Datenbankobjekt ist, können dann anschließend die entsprechenden Rechte an dem `VIEW` vergeben werden.

Beispiel

Betrachten wir dazu ein Beispiel. Der Benutzer `stolze` ist der verantwortliche Kursbetreuer für die Zahlungen der Teilnehmer des Kurses „CE23“. Er soll daher zunächst das Recht bekommen, die Daten aller Kursteilnehmer einzusehen. Um seinen Zugriff auf die Daten des Kurses „CE23“ einzuschränken, wird eine entsprechende `WHERE`-Klausel benötigt. Wir erzeugen dafür wiederum einen `VIEW TeilnehmerCE23` auf die Tabelle `tbKursbesuche`. Anschließend wird dem Benutzer `stolze` dann das Zugriffsrecht auf diesen `VIEW` eingeräumt.

Listing 12.7
Einrichten einer Sicht und der Zugriffsrechte auf diese Sicht

```
CREATE VIEW TeilnehmerCE23 (TeilnehmerID, Zahlung) AS
    SELECT KTID, GezahlterBetrag
        FROM tbKursbesuche
        WHERE KID = 'CE23';
GRANT SELECT
ON TeilnehmerCE23
TO stolze;
```

Tatsächlich spielt diese Vorgehensweise gerade bei größeren Systemen eine wichtige Rolle, da hier vorwiegend auf `VIEW` gearbeitet wird. Der `VIEW` wird als gezielte Sicht für einen Benutzer oder eine Benutzergruppe erzeugt. Daher ist es naheliegend, auch die Berechtigungen für diese Benutzer an den `VIEW` zu knüpfen.

Am Beispiel von MySQL können Sie auch gut sehen, wie einzelne SQL-Befehle in der praktischen Datenbankentwicklung gerade in Hinsicht auf die Berechtigungen erweitert wurden.

CREATE [OR REPLACE]

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]

[DEFINER = { *benutzername* | *CURRENT_USER* }]

[SQL SECURITY { DEFINER | INVOKER }]

VIEW *viewname [(feldnamenliste)]*

AS *SELECT-Anweisung*

[WITH [CASCADED | LOCAL] CHECK OPTION]

Ein `VIEW` kann mit dem Zusatz `SQL SECRITY` entweder unter dem Benutzernamen desjenigen Benutzers ablaufen, der den `VIEW` definiert hat (`DEFINER`), oder unter dem Namen desjenigen, der ihn verwendet, also direkt oder indirekt aufruft (`INVOKER`). Der Name des `DEFINER` ist ohne weitere Angabe der `CURRENT_USER`, der dann als Standardwert verwendet wird. Sie können aber auch einen anderen Benutzernamen bei der Definition angeben, sofern Sie dazu die Rechte haben.

13

Transaktionen

13.1 Grundlagen

Bisher sind wir immer davon ausgegangen, dass eine SQL-Anweisung wie ein Befehl komplett ausgeführt wird. Sie wird eingegeben und endet mit der Bestätigung der Ausführung und der Anzeige des Ergebnisses. Damit sind alle Aktionen abgeschlossen und lassen sich nicht mehr rückgängig machen.

Tatsächlich haben wir schon in Abschnitt 2.3 die Verarbeitung einer SQL-Anweisung etwas ausführlicher untersucht und gesehen, dass sie intern in eine Folge von einzelnen Lese- und/oder Schreibbefehlen zerlegt wird. Was passiert jetzt aber, wenn die Datenbank, während diese Folge abgearbeitet wird, feststellt, dass eine einzelne Anweisung nicht ausgeführt werden kann?

Bisher sind wir auch immer davon ausgegangen, dass wir allein auf die Datenbank zugreifen, wenn eine SQL-Anweisung ausgeführt wird. Zumindest wenn Sie eine der Beispieldatenbanken bei sich installiert haben, war das auch tatsächlich so. Im Regelfall arbeiten aber mehrere Benutzer gleichzeitig auf einer Datenbank, deren Aktionen sich gegenseitig beeinflussen könnten. Es muss dann sichergestellt werden, dass die Befehlsfolge Ihrer SQL-Anweisung nicht durch die Folgen anderer Benutzer „mittendrin“ beeinflusst wird, oder Sie die Anweisung anderer Benutzer beeinflussen.

Transaktionsmanager

Die Lösung für derartige Probleme bietet der Transaktionsmanager, der eine SQL-Anweisung als Transaktion ausführen kann, also eine in sich geschlossene Einheit von Befehlen, die garantiert entweder vollständig oder überhaupt nicht ausgeführt wird. In einem Logbuch werden alle Folgen der Transaktion protokolliert und „zwischengespeichert“. Tritt in der Befehlsfolge ein nicht behebbares Problem auf, das einen erfolgreichen Abschluss der Transaktion verhindert, wird mithilfe des Recoverymanagers das Logbuch genutzt, um alle Folgen der Transaktion zurückzunehmen und den Zustand vor Beginn der Transaktion wiederherzustellen.

Beispiel

Das Problem kann sich innerhalb einer SQL-Anweisung aber auch über mehrere SQL-Anweisungen hinweg erstrecken, in dem Sinn, dass nur die Ausführung aller SQL-Anweisungen zusammen wieder einen sinnvollen Zustand herstellt. Stellen Sie sich vor, wir wollen eine neue Bestellung eintragen, aber sicherstellen, dass sie nur eingetragen wird, wenn der Kunde eingetragen werden kann und die bestellten Produkte verfügbar sind. Der Kunde soll andererseits nur eingetragen werden, wenn seine Bestellung erfasst werden kann. Sie benötigen zumindest eine `INSERT`-Anweisung für den Kunden, eine `INSERT`-Anweisung für die Bestellung und eventuell noch weitere Anweisungen, um die Bestellpositionen einzutragen und die Verfügbarkeit der Artikel zu prüfen.

Gerade bei Änderungen in der Datenbank hängen oft verschiedene Eintragungen und damit verschiedene SQL-Anweisungen voneinander ab. Nur wenn alle Änderungen durchgeführt werden können, ergibt sich wieder ein konsistenter, in sich sinnvoller Stand. Daraus ergeben sich drei wesentliche Anforderungen an eine Transaktion:

1. Die Änderungen müssen so lange rückgängig gemacht werden können, wie nicht sichergestellt ist, dass alle Anweisungen durchführbar sind.
2. Andere Nutzer der Datenbank sollen keine inkonsistenten Zustände sehen können. Die Änderungen dürfen also erst dann für andere Anwender sichtbar werden, wenn sie komplett ausgeführt werden konnten.
3. Änderungen eines Benutzers beziehen sich stets auf den ursprünglichen Zustand der Datenbank bei Beginn der Transaktion. Es können niemals zwei oder mehrere Benutzer gleichzeitig Daten in demselben Bereich der Datenbank ändern. Eine solche Situation könnte abhängig von der Reihenfolge der Änderungen zu für alle Benutzer unerwarteten Situationen führen.

Um diese Anforderungen erfüllen zu können, ist das Konzept der Transaktion eingeführt worden. Eine Transaktion ist eine in sich abgeschlossene Folge von Änderungen und Abfragen, die zunächst so lange gekapselt und vor anderen Benutzern verborgen werden, bis sie als Ganzes entweder durchgeführt (`COMMIT`) oder zurückgenommen (`ROLLBACK`) werden.

Die Unterstützung von Transaktionen in den einzelnen Datenbankmanagementsystemen ist sehr unterschiedlich und hängt teilweise auch von den verwendeten Dateiverwaltungssystemen ab, also den Systemen, die tatsächlich für die physikalische Datenhaltung verwendet werden.

START TRANSACTION

Generell beginnt eine Transaktion entweder automatisch mit der Eingabe einer SQL-Anweisung oder sie muss mit der SQL-Anweisung

```
START TRANSACTION;
```

gestartet werden.

COMMIT

Danach werden alle weiteren SQL-Befehle in einem geschützten Zustand ausgeführt, bis die Folge der Befehle mit

```
COMMIT;
```

Beispiel

(oder `ROLLBACK;`) beendet wird.

Wir wollen einen neuen Kursteilnehmer in die Kursdatenbank aufnehmen. Das könnte etwa mit der INSERT-Anweisung aus Listing 13.1 geschehen.

Listing 13.1
Eintrag eines neuen Kursteilnehmers

```
INSERT INTO tbPerson VALUES(
   99,
   'Klever',
   'Klaus',
   '29223',
   'Celle',
   'Mühlenweg 2',
   '1981-04-02'
);
```

Vor dem COMMIT ist der Befehl noch nicht dauerhaft in der Datenbank gespeichert. Wenn ein anderer Benutzer der Datenbank jetzt mit einem SELECT eine Übersicht über die Personen erstellt, wird der neue Datensatz im Normalfall noch nicht sichtbar. Dies liegt daran, dass noch nicht klar ist, ob die Transaktion vollständig beendet werden kann, ob also nicht noch weitere SQL-Anweisungen auszuführen sind, beispielsweise ein Eintrag in der Tabelle tbKursbesuche. Die Transaktion kann daher jetzt noch mit einem ROLLBACK statt einem COMMIT beendet werden und damit komplett rückgängig gemacht werden. „Klaus Klever“ hat es nie wirklich in der Datenbank gegeben.

ROLLBACK

Tritt während der Transaktion also ein Fehler auf, der nicht behoben werden kann, oder dauert die Transaktion zu lange oder soll sie aus anderen Gründen vom Benutzer ohne Veränderung beendet werden, kann sie mit einem

```
ROLLBACK;
```

komplett rückgängig gemacht werden. In diesem Fall wird der Zustand vor der Durchführung wiederhergestellt.

Wenn Sie nach einem ROLLBACK jetzt ein

```
SELECT * from tbPerson;
```

eingeben, lässt sich das leicht überprüfen. Dies gilt allerdings nur, wenn Ihr Datenbanksystem nicht in einer Art AUTOCOMMIT-Modus läuft, bei dem nach jeder SQL-Anweisung automatisch ein COMMIT ausgeführt wird, wenn diese erfolgreich abgeschlossen werden konnte.

13.2 AUTOCOMMIT

Wenn Sie Transaktionen selbst testen wollen, müssen Sie beachten, dass die meisten Datenbanken in den hier verwendeten Oberflächen im AUTOCOMMIT-Modus laufen. Dies bedeutet, dass jede SQL-Anweisung automatisch als Transaktion mit START TRANSACTION und COMMIT ausgeführt wird. Diesen Modus müssen Sie ausschalten, wenn Sie selbst Transaktionen starten und beenden wollen.

MySQL

MySQL nutzt verschiedene Datenverwaltungssysteme für die physische Datenspeicherung. Nur mit *InnoDB*, *BDB* oder *NDB Cluster* als Dateiverwaltungssystem sind Transaktionen möglich, insbesondere bei Verwendung des

Standards *MyISAM* gilt dies nicht. Daher haben wir bei der Installation *InnoDB* als Standardsystem verwendet. Geben Sie jetzt einfach

```
SET AUTOCOMMIT = 0;
```

an, um AUTOCOMMIT auszuschalten. Entsprechend können Sie es auf 1 setzen, um es wieder einzuschalten.

Oracle

Oracle bietet eine umfangreiche Unterstützung für Transaktionen, die aber je nach Umfang der Version stark variieren. Die einfachste Art für unsere Zwecke ist, das Kontrollkästchen AUTOCOMMIT unmittelbar über dem Eingabefenster der SQL-Befehle zu deaktivieren. Oracle benötigt kein START TRANSACTION. Transaktionen können mit einem COMMIT oder ROLLBACK beendet werden. Das standardkonforme COMMIT WORK ist ebenfalls verfügbar. Es hat denselben Effekt wie ein einfaches COMMIT.

Firebird

Firebird bietet einen vollständigen Support für Transaktionen. Da die Datenbank vorwiegend für den Einsatz als Backend-Datenbanksystem für programmierte Anwendungen gedacht ist, werden Beginn und Ende der Transaktion von der Anwendung gesteuert.

In unserer Oberfläche stehen zwei Schaltflächen zur Verfügung, um ein COMMIT oder ein ROLLBACK der Anweisungen zu steuern.

MS Access

MS Access bietet trotz des ursprünglich als Desktop-System entwickelten Konzeptes eine Transaktionsunterstützung. Eine Transaktion muss aber in jedem Fall mit einem

```
BEGIN TRANSACTION;
```

begonnen werden. Sie kann dann mit einem

```
COMMIT;
```

wieder beendet werden. Statt des einfachen COMMIT kann auch COMMIT TRANSACTION oder COMMIT WORK verwendet werden. Entsprechend gibt es neben dem

```
ROLLBACK;
```

auch ein ROLLBACK TRANSACTION und ein ROLLBACK WORK.

openBase

openBase unterstützt in der aktuellen Version noch keine nennenswerten Transaktionen. Daher sollte hier entweder auf ein anderes System ausgewichen werden oder auf eine spätere Version gewartet werden. Eine Unterstützung ist angekündigt, aber noch nicht terminiert.

13.3 Eigenschaften einer Transaktion

Eine Transaktion besteht also im Prinzip aus folgender Struktur:

Transaktionsstruktur

[START TRANSACTION]

SQL-Befehle

[COMMIT|ROLLBACK]

In einer Transaktion können theoretisch beliebig viele SQL-Befehle aneinandergereiht werden. Aber auch bei einem einzigen Befehl kann dies hilfreich sein. Kommen wir dazu auf unser Kursbeispiel zurück. Es sollen Rechnungen für die Kursteilnehmer erstellt werden:

```
UPDATE artikel SET Listenpreis = Listenpreis * 1.03;
```

Das Update der Artikel wird in realen Datenbeständen zumeist nur wenige Sekunden oder Sekundenbruchteile benötigen. Trotzdem kann es in dieser Zeit zu Störungen kommen. Ein Systemabsturz, ein Stromausfall, ein Plattenfehler oder verschiedene andere mögliche Störungen könnten auftreten. In diesem Fall entsteht ein undefinierter Zustand, bei dem zunächst mühsam geklärt werden muss, welche Daten aktualisiert wurden und welche nicht.

Daher werden wann immer möglich Transaktionen eingesetzt. Dies gilt in besonderem Maß für Befehle, die den Inhalt der Datenbank ändern.

ACID

Die Eigenschaften, die eine Transaktion ausmachen, wurden von der Datenbanktheorie ursprünglich unter der Abkürzung *ACID* definiert. ACID steht dabei für die Anfangsbuchstaben der englischen Begriffe für die Eigenschaften Atomicity, Consistency, Isolation und Durability. In der deutschen Sprachwahl haben sich weitgehend die Begriffe Atomarität, Konsistenz, Isolation und Dauerhaftigkeit durchgesetzt.

- Atomarität – Eine Transaktion ist ein unteilbares Ganzes, sie wird entweder komplett durchgeführt (`COMMIT`) oder komplett rückgängig gemacht (`ROLLBACK`).
- **Konsistenz** – Eine Datenbank, die in einem in sich konsistenten Zustand ist, ist auch nach Abschluss der Transaktion wieder in einem konsistenten Zustand. Durch die vollständige Durchführung wird gewährleistet, dass keinerlei Verstöße gegen Konsistenzbedingungen möglich sind, soweit dies in der Kontrolle der Datenbank liegt.
- **Isolation** – Die SQL-Anweisungen innerhalb einer Transaktion werden nicht von SQL-Anweisungen anderer Transaktionen beeinflusst und umgekehrt. Wie wir noch sehen werden, können Abweichungen durch weniger restriktive Isolationsstufen ermöglicht werden.
- **Dauerhaftigkeit** – Bis zum Abschluss einer Transaktion können alle Befehle rückgängig gemacht werden. Mit Abschluss der Transaktion sind die durch die Transaktion ausgelösten Änderungen dauerhaft in der Datenbank gespeichert.

13.4 Sperren

Jede Transaktion stellt sicher, dass ein Benutzer Änderungen vornehmen kann, ohne dass ein anderer Benutzer die von der Änderung betroffenen Bereiche gleichzeitig ändern kann. Dazu sind sogenannte Sperren notwendig.

Sperren sind Bereiche der Datenbank, die – zumeist kurzzeitig – für einen Benutzer zur alleinigen Verwendung reserviert werden und für alle anderen Benutzer nicht mehr zugreifbar sind. Das hat zwar für den einen Benutzer den Vorteil der störungsfreien Änderung der Datenbank, für alle anderen Benutzer allerdings den Nachteil, dass bestimmte Bereiche der Datenbank – zumindest weitgehend – nicht mehr zugreifbar sind.

Ein solcher Zustand behindert natürlich den Betrieb der Datenbank und muss auf ein Mindestmaß reduziert werden. Mindestmaß bedeutet dabei zeitlich, dass die Transaktion auf die unbedingt notwendige Zeit beschränkt wird. Mindestmaß bedeutet auch, dass die anderen Benutzer nicht mehr als unbedingt notwendig eingeschränkt werden. Dies wird durch die Sperrstrategie bestimmt.

Sperrstrategie

Sperrstrategie

Bei der Sperrstrategie wird zunächst nach Sperren auf Satzebene und Sperren von größeren Einheiten, zumeist ganzen Tabellen oder Teilen beispielsweise Seiten im Sinne der physischen Datenspeicherung, unterschieden.

Sperre auf Satzebene (Record)

Sperren auf Satzebene blockieren nur einen kleinen Bereich, da nur die von Änderungen unmittelbar betroffenen Datensätze gesperrt werden. Andererseits entsteht ein erheblicher Verwaltungsaufwand und gerade `SELECT`-Anweisungen, die größere Bereiche einer Tabelle durchsuchen müssen, können trotzdem behindert werden. Generell ist diese Strategie vorteilhaft, wenn verglichen mit den `SELECT`-Abfragen sehr viele Änderungen an einer Datenbank passieren, da sich verschiedene `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen, die auf Satzebene ablaufen, vergleichsweise wenig behindern.

Sperre auf Tabellenebene

Die entgegengesetzte Strategie – Sperren auf Tabellenebene – beruht darauf, dass bereits die Änderung eines Datensatzes zum Sperren der gesamten Tabelle führt. Dies scheint einerseits übertrieben, erfordert aber andererseits wenig Verwaltungsaufwand. Werden Tabellen nur selten geändert – wie beispielsweise bei den meisten Internetanwendungen – ist dies eine sehr effektive Vorgehensweise.

Unterschiedliche Typen von Sperren

Eine weitere Möglichkeit, die Behinderungen anderer Anwender einzuschränken, besteht darin, unterschiedliche Sperrtypen zu verwenden. Im Allgemeinen werden exklusive Sperren und nicht-exklusive Sperren – Shared-Sperren – unterschieden.

Exklusive Sperre

Bei einer exklusiven Sperre wird der gesperrte Bereich – Datensatz, Tabelle oder Seite – für einen Benutzer exklusiv gesperrt. Er kann von ihm geändert oder gelöscht werden. Transaktionen anderer Benutzer erhalten keinen Zugriff und müssen mit Schreib- und Löschoperationen warten. Eine exklusive Sperre wird durch jede ändernde SQL-Anweisung ausgelöst, insbesondere durch `UPDATE` und `DELETE`. Sie kann auch durch eine `SELECT`-Anweisung mit dem Zusatz `FOR UPDATE` bewusst ausgelöst werden:

```
START TRANSACTION
SELECT * FROM artikel WHERE anr = 6001 FOR UPDATE;
UPDATE artikel SET einstandspreis = 1.64 WHERE anr = 6001;
COMMIT;
```

Listing 13.2
Exklusive Sperre im SELECT mit anschließendem Update

Der Sinn dieser Transaktion besteht darin, bereits bei der Abfrage die betroffenen Datensätze oder die gesamte Tabelle zu sperren. Ist die Abfrage dabei erfolgreich, können die folgenden Änderungen in jedem Fall durchgeführt werden. Kein anderer Benutzter sieht die ersten Änderungen, solange nicht alle Preisänderungen komplett durchgeführt wurden.

Nicht exklusive Sperre

Bei einer nicht-exklusiven Sperre wird der gesperrte Bereich nur als „Shared Read-Bereich" gespeichert. Die Transaktion kann also davon ausgehen, dass keine andere Transaktion die gelesenen Daten ändern oder löschen, wohl aber lesen kann. Der Vorteil liegt darin, dass beispielsweise zunächst geprüft werden kann, ob ein Eintrag in der Artikeltabelle vorhanden ist. Erst wenn dies der Fall ist, wird die zugehörige Bestellung eingefügt. Da die Tabelle `artikel` nicht geändert werden soll, besteht auch kein Grund, andere Transaktionen am Lesen zu hindern. Der Eintrag in der Artikeltabelle soll nur nicht während der Eintragung der Bestellung geändert oder gelöscht werden können.

```
START TRANSACTION;
SELECT * FROM artikel WHERE anr = 6001 LOCK IN SHARE MODE;
INSERT INTO bestellung VALUES (1,4919,6,'2007-11-
22',CURRENT_TIMESTAMP);
INSERT INTO bestell_position VALUES
(1,4919,1,6001,10,CURRENT_TIMESTAMP);
COMMIT;
```

Listing 13.3
Einfügen zweier Datensätze nach einer nicht-exklusiven Sperre

Wir haben bereits gesehen, dass sich ähnliche Konsistenzprüfungen über die referenzielle Integrität sicherstellen lassen. Die Datenbank weiß um den Zusammenhang und führt die entsprechenden Sperren selbstständig durch. Soll dies nicht erfolgen, kann die referenzielle Integrität also auch über eine entsprechende Transaktion, wie in Listing 13.3 zu sehen, sichergestellt werden.

Der Vorteil dieser Sperre ist, dass während der Transaktion jederzeit andere Transaktionen ebenfalls `SHARED READ`-Sperren errichten können. Damit können mehrere Transaktionen parallel sicherstellen, dass die mit `LOCK IN SHARE MODE` abgefragten Daten nicht von anderen Transaktionen geändert oder gelöscht werden können. Natürlich kann keine Transaktion auf einem solchen Bereich eine exklusive Sperre zum Ändern oder Löschen errichten. Diese Transaktion muss warten, bis alle anderen Transaktionen beendet wurden und ihre Sperren entsprechend aufgehoben wurden.

In der Praxis finden sich noch eine Reihe weiterer Sperren, die Sie der Dokumentation des jeweiligen Datenbankmanagementsystems entnehmen können.

Isolationslevel von Transaktionen

Isolationslevel

Hinzu kommt der Isolationslevel der Transaktion. Grundsätzlich sieht der SQL-Standard vier Ebenen vor, die eine steigende Trennung der Transaktion von der restlichen Datenbank beinhalten, die Isolationslevel. Im Wesentlichen beeinflusst der Isolationslevel die Art wie geänderte Informationen für die Transaktion selbst und für andere Transaktionen sichtbar werden.

Die wesentlichen Isolationslevel sind:

- READ UNCOMITTED
- READ COMITTED
- READ REPEATABLE
- SERIALIZABLE

Der Isolationslevel kann entweder als Standard in der Konfiguration des Datenbankmanagementsystems eingestellt oder beim Start einer Transaktion angegeben werden. In den meisten Systemen steht dazu die SQL-Anweisung

```
SET TRANSACTION;
```

zur Verfügung, die neben dem Isolationslevel zumeist die Einstellung einer ganzen Reihe weiterer Eigenschaften einer Transaktion erlaubt. In anderen Fällen wie in MySQL ist es auch möglich, den Isolationslevel bereits beim Start der Transaktion anzugeben. Im Folgenden werden einige Beispiele mit MySQL vorgestellt. Dabei werden stets zwei „Benutzer“ simuliert, die mit ihren Transaktionen zugreifen.

READ UNCOMMITTED

Die schwächste Isolationsstufe ist der `READ UNCOMMITTED`, der bei einem `SELECT` auch bereits das Lesen von Daten anderer nicht beendeter Transaktionen erlaubt. Dies wird auch als „schmutziges Lesen“ (dirty read) bezeichnet, da auch Daten gelesen werden, die unter Umständen nicht konsistent sind. Bei einem `ROLLBACK` der anderen Transaktion, deren Daten gelesen werden, verschwinden diese wieder und es wurden Daten gelesen, die nie Bestandteil des offiziellen Datenbankstandes waren und sind.

Tabelle 13.1 Transaktion mit Isolationsstufe READ UNCOMMITED

Eigene Transaktion (T1)	Fremde Transaktion (T2)
`START TRANSACTION`	
`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`	
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen 1 Bestellung des Kunden 1.	
	`START TRANSACTION ...`
	`SELECT * FROM bestellung WHERE KID = 1 LOCK IN SHARE MODE;` 1 Datensatz des Kunden gefunden.

Eigene Transaktion (T1)	Fremde Transaktion (T2)
	`INSERT INTO bestellung VALUES (1,9999,1, CURRENT_DATE, CURRENT_TIMESTAMP);` neue Bestellung von KID=1
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen die neue Bestellung des Kunden 1.	
	`ROLLBACK;`
`SELECT * FROM bestellung WHERE KID = 1;` Die neue Bestellung ist wieder verschwunden. Sie war nie wirklich Bestandteil der Datenbank. Sie sehen nur noch eine Bestellung des Kunden 1.	
`COMMIT;`	

Tabelle 13.1 (Forts.) Transaktion mit Isolationsstufe READ UNCOMMITED

Der Vorteil liegt hier in dem geringen Verwaltungsaufwand der Datenbank. Der Nachteil ist, dass Daten wie Phantome auftauchen können, da sie von einer anderen Transaktion zwischenzeitlich eingepflegt waren, die dann aber wieder rückgängig gemacht wurde. Damit können Abfragen Ergebnisse liefern, die in großen Mehrbenutzersystemen nie wieder nachvollziehbar oder erklärbar sind.

READ COMMITTED

Die nächste Isolationsstufe ist der `READ COMMITED`, die Standardeinstellung von Oracle, die bei einem `SELECT` nur diejenigen Daten anderer Transaktionen liest, die bereits durch einen `COMMIT` abgeschlossen wurden. Die eigenen aktualisierten Daten werden gelesen.

Eigene Transaktion (T1)	Fremde Transaktion (T2)
`START TRANSACTION`	
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen 1 Bestellung des Kunden 1.	
	`START TRANSACTION`
	`SELECT * FROM bestellung WHERE KID = 1 LOCK IN SHARE MODE;` 1 Datensatz des Kunden gefunden;
	`INSERT INTO bestellung VALUES (1,9999,1, CURRENT_DATE, CURRENT_TIMESTAMP);`

Tabelle 13.2 Transaktion mit der Isolationsstufe READ COMMITTED

Tabelle 13.2 (Forts.) Transaktion mit der Isolationsstufe READ COMMITTED

Eigene Transaktion (T1)	Fremde Transaktion (T2)
	`SELECT * FROM bestellung WHERE KID = 1 LOCK IN SHARE MODE;` Sie sehen auch die neue Bestellung von KID=1.
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen nach wie vor nur die erste Bestellung des Kunden mit der KID=1.	
	`COMMIT;`
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen auch die neue Bestellung von KID=1.	
`COMMIT;`	
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen auch die neue Bestellung von KID=1.	

Sie sehen, dass die Änderungen der fremden Transaktion erst sichtbar werden, wenn diese mit einem `COMMIT` beendet wurde. Das Ergebnis der Abfrage in der eigenen Transaktion ist unabhängig von einem `COMMIT` der eigenen Transaktion.

Die nächste Isolationsstufe `REPEATABLE READ` ist die Standardeinstellung von MySQL. Im Unterschied zu `READ COMMITED` werden bei einem `SELECT` jetzt stets die Daten gelesen, die beim Beginn der eigenen Transaktion aktuell waren.

Tabelle 13.3 Transaktion mit dem Isolationslevel REPEATABLE READ

Eigene Transaktion (T1)	Fremde Transaktion (T2)
`START TRANSACTION`	
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen 1 Bestellung des Kunden 1.	
	`START TRANSACTION`
	`SELECT * FROM bestellung WHERE KID = 1 LOCK IN SHARE MODE;` 1 Datensatz des Kunden gefunden.
	`INSERT INTO bestellung VALUES (1,9999,1, CURRENT_DATE, CURRENT_TIMESTAMP);`

Eigene Transaktion (T1)	Fremde Transaktion (T2)
	`SELECT * FROM bestellung WHERE KID = 1 LOCK IN SHARE MODE;` Sie sehen auch die neue Bestellung von KID=1.
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen nach wie vor nur die erste Bestellung des Kunden mit der KID=1.	
	`COMMIT;`
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen nach wie vor nur die erste Bestellung des Kunden mit der KID=1.	
`COMMIT;`	
`SELECT * FROM bestellung WHERE KID = 1;` Sie sehen auch die neue Bestellung von KID=1.	

Tabelle 13.3 (Forts.) Transaktion mit dem Isolationslevel REPEATABLE READ

Die vierte und letzte definierte Stufe ist `SERIALIZABLE`. Bei `SERIALIZABLE` wird die Einstellung des `READ COMMITTED` mit dem Unterschied verwendet, dass jedes `SELECT` automatisch eine `SHARED MODE`-Sperre setzt und so sicherstellt, dass tatsächlich keine Änderungen an den Daten mehr vorgenommen werden können. Dies stellt sicher, dass Daten, die einmal abgefragt wurden, bei einer folgenden Änderung in derselben Transaktion mit Sicherheit in unverändertem Zustand vorliegen.

13.5 Transaktionen und Benutzervariablen

Eine weitere Nutzungsmöglichkeit von Transaktionen ergibt sich im Zusammenhang mit Benutzervariablen. Sollen Zwischenergebnisse gespeichert werden, so können diese in Variablen abgelegt und dann innerhalb einer Transaktion weiterverwendet werden.

Beispiel

Es soll eine Liste aller Personen ausgegeben werden, die zusätzlich eine Zeilennummerierung besitzen soll. Dann kann in MySQL eine lokale Variable `@Zeilennummer` definiert werden, die anschließend in einer `SELECT`-Anweisung genutzt wird.

```
START TRANSACTION;
SET @Zeilennummer = 0;
SELECT @Zeilennummer := @Zeilennummer +1 AS "Zeile", PID FROM tbPer-
son;
COMMIT;
```

Listing 13.4 Hinzufügen einer Zeilennummerierung zu einer Abfrage

Das Listing 13.4 zeigt, wie innerhalb einer Transaktion zunächst der Wert definiert wird, der anschließend in der SELECT-Anweisung wie in einer Schleife durchlaufen wird. Grundsätzlich lassen sich auch globale Variablen verwenden, die aber wegen der Seiteneffekte sehr vorsichtig zu verwenden sind. Der hier beschriebene Mechanismus bleibt lokal und setzt die Verwendung einer Transaktion voraus, da nur innerhalb der Transaktion der Kontext erhalten bleibt.

Sie können dies testen, indem Sie die entsprechenden Befehle nacheinander ausführen lassen.

14

Mit SQL Datenbanken betreiben und optimieren

Die Data Control Language (DCL) bietet innerhalb von SQL neben den Berechtigungen und Transaktionen noch eine Reihe von Unterstützungen bei der physischen Datenspeicherung. Nachdem die physische Datenhaltung bei allen Datenbanksystemen unterschiedlich ist, gibt es letztlich auch Unterschiede in der DCL. Daher sollen hier abschließend noch einige Aspekte der DCL im Überblick erläutert werden, ohne auf die Details der einzelnen Systeme umfassend eingehen zu können.

14.1 Optimierter Zugriff – der INDEX

14.1.1 Nutzen von Indizes

„... da müssen wir noch einen Index drauflegen ...“ ist einer der gängigsten Sätze im Datenbankmanagement. Der Index ist in vielen Fällen das Allheilmittel, um schlechte Zugriffszeiten auf die Datenbank zu optimieren. Ein Index kann in vielen Situationen, den Ablauf erheblich beschleunigen, beispielsweise:

- die Suche nach bestimmten Datensätzen (`WHERE`-Klausel)
- die Sortierung von Tabellen nach bestimmten Attributwerten (`ORDER BY`-Klausel)
- die Gruppierung nach Attributen (`GROUP BY`- und `HAVING`-Klausel)
- die Abfrage über mehrere Tabellen hinweg (Fremdschlüssel-Primärschlüssel-Beziehungen)

Ein Index ist immer ein Zusatz zu einer Tabelle, sozusagen eine eigene zusätzliche „Tabelle“. Dieser Index bezieht sich auf ein oder mehrere Datenfelder der zugrunde liegenden Tabelle. Diese Indextabellen sind kleiner und die Werte dieser Datenfelder werden für alle Datensätze neu angeordnet,

sodass ein schnellerer Zugriff erfolgen kann. Jeder Eintrag in der Indextabelle verweist dann auf den Originaldatensatz in der Originaltabelle.

Geringerer Speicherplatz

Da im Rahmen des Index nur die Werte der Indexfelder berücksichtigt werden, wird relativ wenig Speicherplatz für den Index benötigt. Durch die geringe Größe der Information kann wesentlich mehr Information mit einem Zugriff gelesen werden und es können unter Umständen schnellere Speicher (Cache) für die Speicherung eines Index verwendet werden. Dies zusammen mit der für die Suche optimierten Struktur ermöglicht eine wesentlich schnellere Suche nach Datensätzen als in den Originaltabellen. In einem Index kann gesucht werden und anschließend mithilfe des Querverweises auf die kompletten gefundenen Datensätze unmittelbar zugegriffen werden. Die Verwaltung des Index und die Suche sind ausschließlich Sache der Datenbank. Sie erstellen Ihre SQL-Anweisung – insbesondere die `SELECT`-Anweisung – und das Datenbankmanagementsystem optimiert die Abfrage, indem es die infrage kommenden Indizes berücksichtigt.

Verwendung sichern

Wie gesagt, Sie müssen sich theoretisch nicht um die Verwendung der von Ihnen angelegten Indizes in den `SELECT`-Anweisungen kümmern, praktisch ist es allerdings ratsam, gerade für häufig eingesetzte `SELECT`-Anweisungen eine Analyse der verwendeten Indizes vorzunehmen. Dies stellt sicher, dass die gewünschten Indizes tatsächlich verwendet werden. Das ist nicht so selbstverständlich, wie es sich zunächst anhört, praktisch können bereits kleine Unterschiede in einer `SELECT`-Anweisung die Nutzung eines Index verhindern und so zu erheblich längeren Antwortzeiten und einer erheblich größeren Belastung des Datenbankmanagementsystems führen.

Bevor wir weitere Überlegungen zum Einsatz von Indizes machen, soll jetzt auf die konkrete SQL-Syntax für die Verwaltung von Indizes eingegangen werden.

14.1.2 Einen Index anlegen (CREATE INDEX)

Beispiel

In der Kursdatenbank besteht beispielsweise der Primärschlüssel der Tabelle `tbPerson` aus der `PID`. Wer kennt aber immer die `PID`? So wird man beim Zugriff zwar seinen Namen, aber nicht seine `PID` wissen. Daher ist es naheliegend, dass in der Kursdatenbank eine Person oft mit dem Namen gesucht werden muss. Der passende Index könnte (in MySQL) wie folgt erzeugt werden:

```
CREATE INDEX name
ON tbPerson (Familienname);
```

Ein Index hat immer einen eigenen Namen, wie hier `name`, und bezieht sich immer auf eine Tabelle. Er kann ein oder mehrere Datenfelder der Tabelle umfassen. Hier wird ein Index auf dem Feld `Familienname` der Tabelle `tbPerson` definiert. Für jede Tabelle können kein, ein oder mehrere Indizes definiert werden. Da sich ein Index immer auf eine Tabelle bezieht, kann er auch direkt mit der Erstellung der Tabelle definiert werden, also im Zusammenhang mit der `CREATE TABLE`-Anweisung.

Index-Erstellung mit CREATE TABLE

CREATE TABLE (

[Felddefinition|Integritätsbedingung] { ,[Felddefinition|Integritätsbedingung]};

Als Integritätsbedingung kann dabei auch ein

INDEX [*indexname] (feldname [(Präfixlänge)] { ,feldname [(Präfixlänge)] })*

verwendet werden. Der Index erhält dabei wiederum einen eigenen Namen, da es sich um ein eigenständiges Datenbankobjekt handelt. Im Anschluss an den Indexnamen wird eine Liste der Feldnamen angegeben, auf die sich der Index beziehen soll. Je Datensatz werden die Werte dieser Felder berücksichtigt. Da ein Index umso effektiver funktioniert und umso weniger Speicherplatz benötigt, je kleiner er ist, ist es bei langen Textfeldern sinnvoll, die Anzahl der Zeichen je Feld zu begrenzen. So kann beispielsweise bei einem Namensfeld, das 50 Zeichen lang ist, der Index auf die ersten 10 Zeichen (Präfixlänge) beschränkt werden. Dies ist im Normalfall vollkommen ausreichend, um mit einem `SELECT` eine so kleine Menge infrage kommender Datensätze zu ermitteln, dass diese anschließend sequenziell durchsucht werden können.

Index-Erstellung mit ALTER TABLE

Ein Index kann nicht nur im Zusammenhang mit der Erstellung einer neuen Tabelle angelegt werden, sondern er kann jederzeit auch nachträglich eingefügt werden. Nachdem er wie eine Integritätsbedingung behandelt wird, kann dies auch mit der bereits bekannten `ALTER TABLE`-Anweisung geschehen, die für jede Art von Änderung der Tabellendefinition genutzt werden kann.

ALTER TABLE *tabellenname ADD INDEX indexname (feldname [(Präfixlänge)]*

{ ,feldname [(Präfixlänge)] });

Oder konkret:

ALTER TABLE tbPerson ADD INDEX name (Familienname(10));

Die Bedeutung der Indizes erkennen Sie auch daran, dass die meisten Datenbankmanagementsysteme eine eigene SQL-Anweisung zur Erstellung von Indizes entwickelt haben. Diese bereits im anfänglichen Beispiel verwendete SQL-Anweisung `CREATE INDEX` besitzt eine große Bedeutung.

CREATE INDEX

CREATE INDEX *indexname ON tabellenname (feldname[(Präfixlänge)]*

{ ,feldname[(Präfixlänge)] });

Die Syntax entspricht im Wesentlichen der Syntax der `ALTER TABLE`-Anweisung und die meisten Datenbankmanagementsysteme, so auch MySQL, bilden die `CREATE INDEX`-Anweisung intern auf einer `ALTER TABLE`-Anweisung ab und führen diese aus.

Generell ist es ratsam, den Index mit einer ALTER TABLE- oder CREATE INDEX-Anweisung zu erstellen. Hier werden Sie gezwungen, einen Namen für den Index zu vergeben, der für die spätere Analyse oder das Löschen des Index hilfreich ist.

Kommen wir auf das obige Beispiel zurück.

Listing 14.1 Abfrage mit Nutzung eines Index

```
CREATE INDEX name
ON tbPerson (Familienname);

SELECT *
FROM tbPerson p
WHERE p.Familienname = 'Weiss';
```

Im Listing 14.1 werden zwei SQL-Anweisungen verwendet. Die SELECT-Anweisung sollte theoretisch nach der Einrichtung eines Index schneller ablaufen. Die tatsächlichen Ausführungszeiten hängen natürlich von der Hardware, dem Betriebssystem und den sonstigen aktiven Prozessen ab. Beispielhaft könnte man mit MySQL die Zeiten in der Tabelle 14.1 ermitteln.

Tabelle 14.1 Zugriffszeiten

	Reiner DB-Zugriff	Zugriff mit Aufbereitung
ohne Index	0,0011s	0,0454s
mit Index	0,0331s	0,0410s

Die reinen Ausführungszeiten beschreiben die Zeit für die Datenbankabfrage. Die Zeit für die Aufbereitung umfasst zusätzlich die Zeit für die Aufbereitung in der Oberfläche. Sie sehen, dass die Zugriffszeiten durch den Index nicht wirklich besser werden, die reine Datenbankzugriffszeit wird sogar schlechter. Dies hängt mit dem zusätzlichen Verwaltungsaufwand für den Index zusammen. Die Nutzung eines Index lohnt nur bei relativ großen Datenmengen (siehe Tabelle 14.2).

Haben Sie entsprechend große Datenmengen, so kann ein Index lohnen. Sie müssen dann allerdings sicherstellen, den richtigen Index zu verwenden. Je nach Zugriff auf die Kundendaten können verschiedene Indizes sinnvoll sein (immer vorausgesetzt, dass eine entsprechend große Anzahl Datensätze vorhanden ist).

Beispiel

Es soll folgende SQL-Anweisung auf die Personentabelle der Kursdatenbank ausgeführt werden.

Listing 14.2 Beispielabfrage für die Indexverwendung

```
SELECT *
   FROM tbPerson
   WHERE Familienname = 'Weiss' AND Ort = 'Hannover';
```

Es werden verschiedene Indizes für die Tabelle tbPerson definiert, wie in Listing 14.3 angegeben.

Listing 14.3 Drei verschiedene Indizes für die Tabelle tbPerson

```
CREATE INDEX plz ON tbPerson (Familienname, PLZ);
CREATE INDEX name ON tbPerson (Vorname, Familienname, Ort);
CREATE INDEX ort ON tbPerson (Familienname(10), Ort, Vorname);
CREATE INDEX ort2 ON tbPerson (Ort, Familienname(10));
```

Die Frage ist nun, welche dieser Indizes für die SELECT-Anweisung verwendet werden können. Schauen wir uns dazu den Effekt der Anlage eines Index an. Die Tabelle tbPerson ist in Abbildung 14.1 komplett zu sehen, die Reihenfolge der Felder wurde für unsere Zwecke in der Anzeige geändert.

PID	Familienname	Vorname	Ort	PLZ
1	Weiss	Peter	Hannover	30529
2	Bucz	Susanne	Hannover	30531
4	Karmann	Thomas	Celle	29227
5	Klötzer	Karl	Celle	29221
6	Weiss	Karin	Hannover	30529
7	Weiss	Peter	Braunschweig	38134
8	Meier	Kathrin	Braunschweig	38154
9	Schmidt	Karl	Hannover	30529
10	Müller	Claudia	Breitenhees	29596
11	Lisewski	Bernd	Barsinghausen	30890
15	Martens	Melanie	Celle	29221
17	Schlachter	Dieter	Celle	29227
23	Peredy	Helmut	Celle	29221
24	Ruppert	Nicola	Bergen	29301
25	Sander	NULL	Celle	29223
26	Cromberg	Jörg	Braunschweig	38152
31	Schulze	Tanja	Winsen	29308
32	Winter	Petra	Hermannsburg	29320
34	Plate	Ulrich	Hannover	30529
37	Magerkurth	Melissa	Nienhagen	29336

Abbildung 14.1
Inhalt der Tabelle tbPerson

Wird der Index ort, wie er als dritte Zeile in Listing 14.3 angegeben ist, angelegt, kann man sich die Anlage einer Indextabelle wie in Abbildung 14.2 vorstellen. Die Felder des Index werden für jeden Datensatz zu einem Eintrag verbunden und es wird als Referenz die ID des Datensatzes angegeben. Tatsächlich wird nicht die PID, sondern ein interner, nur dem Datenbanksystem bekannter Schlüssel verwendet, um den richtigen Datensatz zu einem Indexeintrag zu finden. Dies kann statt eines einzelnen Datensatzes auch der Name einer Speicherseite oder ein anderer Eintrag sein, der dem Datenbankmanagementsystem einen schnellen Zugriff erlaubt. Hier wird also nur aus Vereinfachungsgründen die PID gewählt, um das Prinzip zu verdeutlichen.

MySQL bildet jetzt aus der WHERE-Bedingung der SELECT-Abfrage ebenfalls einen zusammengesetzten Text, also „WeissHannover“, und beginnt damit, in der Indextabelle zu suchen. Wichtig ist dabei, dass MySQL nur von links vergleichen kann, es kommt also darauf an, dass der Anfang des Suchstrings zum Anfang des Indexeintrags passt. Dies ist bei dem Index ort tatsächlich der Fall, sodass der Index verwendet werden kann, um die PID des oder der gesuchten Datensätze zu ermitteln. Wie gesagt, handelt es sich dabei in der Praxis um die physischen Adressen, sodass der Zugriff dann gezielt erfolgen kann.

Abbildung 14.2
Mögliche Indextabelle zum Index Ort

Indexeintrag	Interne ID
WeissHannover30529	1
BuczHannover30531	2
KarmannCelle29227	4
KlötzerCelle29221	5
WeissHannover30529	6
WeissBraunschweig38134	7
MeierBraunschweig38154	8
SchmidtHannover30529	9
MüllerBreitenhees29596	10
LisewskiBarsinghausen30890	11
MartensCelle29221	15
SchlachterCelle29227	17
PeredyCelle29221	23
RuppertBergen29301	24
SanderCelle29223	25
CrombergBraunschweig38152	26
SchulzeWinsen29308	31
WinterHermannsburg29320	32
PlateHannover30529	34
MagerkurthNienhagen29336	37

Damit zeigt sich, dass für die obige SQL-Abfrage tatsächlich nur der Index `ort` verwendbar ist. Der Index `ort2` enthält zwar die Angaben, allerdings in der falschen Reihenfolge. Hier müsste die Abfrage wie in Listing 14.4 lauten. Ein guter Optimierer erkennt dies selbstständig, darauf verlassen kann man sich allerdings nicht.

Listing 14.4
Beispielabfrage für die Indexverwendung mit dem Index ort2

```
SELECT *
   FROM tbPerson
   WHERE Ort = 'Hannover' AND Familienname = 'Weiss';
```

Der Index `plz` wäre teilweise nutzbar, da zumindest der Familienname als linker Anfangsteil verwendet werden kann. Es können alle Personen mit dem Namen „Weiss“ ermittelt werden. Die Datensätze können ermittelt und dann kann in den Datensätzen geprüft werden, ob diese Personen in Hannover wohnen. Existieren nicht zu viele Personen mit dem Namen „Weiss“, kann dies durchaus noch effizient sein.

Der Index `name` ist dagegen für die SQL-Anweisung überhaupt nicht nutzbar, da der zunächst links benötigte Familienname aus der `WHERE`-Klausel nicht bekannt ist. Damit bleibt hier nur die vollständige Suche in der Tabelle.

Entsprechende Vorsicht ist auch bei der Verwendung der Platzhalter, insbesondere des %-Zeichens, angebracht. Steht dieses links in einer Vergleichsbedingung ist der linke Teil der Vergleichsbedingung nicht bekannt und ein Index ist nicht nutzbar.

```
SELECT * FROM tbPerson WHERE Familienname = '%eiss';
SELECT * FROM tbPerson WHERE Familienname = 'W%';
```

Listing 14.5
Nur die zweite Abfrage erlaubt eine Indexverwendung.

Insofern liegen also bei großen Datenmengen zwischen den beiden Abfragen in Listing 14.5 erhebliche Unterschiede. Nur die zweite Abfrage erlaubt die Verwendung eines Index, während die erste einen kompletten Scan der Tabelle erfordert.

14.2 Einen Index löschen

Ein Index kann nicht nur angelegt werden, er kann natürlich auch wieder entfernt werden.

Wir wollen zunächst den oben definierten Index wieder entfernen:

```
DROP INDEX name ON kunden;
```

Die allgemeine Syntax für das Entfernen eines Index lautet:

DROP INDEX name;

MySQL erfordert eine zusätzliche Angabe

ON *tabellenname*

in der `DROP INDEX`-Anweisung. Bei der Entfernung des Index werden einfach die zusätzlichen Indextabellen gelöscht. Da die Originaltabellen davon nicht berührt sind, kann dies ohne Datenverlust geschehen. Die Anweisung benötigt den Indexnamen, der bei der Erzeugung des Index vergeben wurde.

Die Entfernung des Index kann auch mit der entsprechenden `ALTER TABLE`-Anweisung

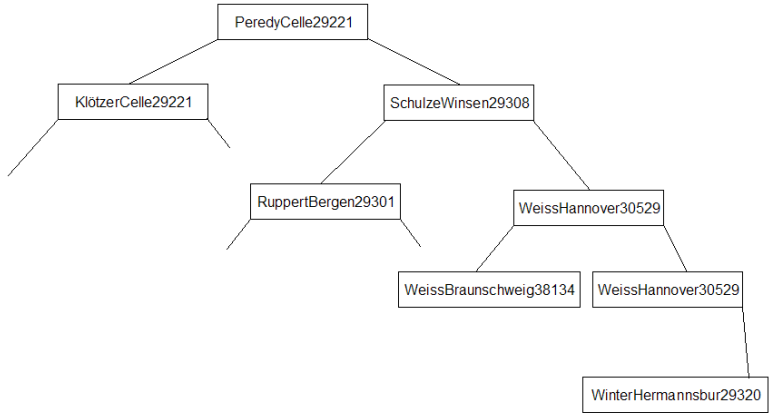
ALTER TABLE *tabellename DROP INDEX indexname;*

erfolgen.

14.3 Weitere Überlegungen zum Einsatz von Indizes

Der Einsatz eines Index lohnt nur bei einer entsprechend großen Menge von Einträgen. Die Informationen eines Index werden zumeist in sogenannten B-Bäumen gespeichert. Ein solcher B-Baum beruht auf den aus der Graphentheorie stammenden Bäumen. Die Indexeinträge eines Baumes sind für unseren Index `ort` ausschnittsweise in Abbildung 14.3 dargestellt.

Abbildung 14.3
Baum für einen Index



Sie sehen, dass jeder Knoten im Baum einem Eintrag der Indextabelle entspricht. Der Eintrag der zugehörigen physischen ID kann dort ebenfalls gespeichert sein. Ist jetzt ein Eintrag wie „WeissHannover“ gesucht, wird oben in der Wurzel des Baums eingestiegen. Durch Vergleich mit dem Eintrag im Baum kann entschieden werden, ob der gesuchte Eintrag links oder rechts im Unterbaum liegen muss, da links alle kleineren und rechts alle größeren Einträge gemäß der verwendeten Sortierung liegen. Aus dem Vergleich

```
WeissHannover > PeredyCelle
```

ist eindeutig entscheidbar, dass der gesuchte Eintrag im rechten Unterbaum liegt. Es wird also danach mit „SchulzeWinsen29308“ verglichen. Wieder muss der gesuchte Eintrag rechts liegen. Hier wird der Eintrag gefunden und anschließend geprüft, ob in den Unterbäumen weitere Einträge vorhanden sind. Der genaue Algorithmus soll hier nicht interessieren. Wichtig ist für die Performance der Abfrage, derartige Bäume möglichst ausgeglichen (balanced) zu gestalten. Dies garantieren die sogenannten B-Bäume mit einem speziellen Einfüge- und Ausgleichsmechanismus.

Bei einem ausgeglichenen Baum lässt sich die Tiefe t aus der Anzahl der Datensätze n mit der Formel

$$t = \log_2 n$$

näherungsweise bestimmen. Die Tiefe des Baums bestimmt wiederum die Anzahl der Datenzugriffe und Vergleiche, die notwendig sind, um einen bestimmten Datensatz zu finden. Während bei einer Suche ohne Index sequenziell alle Datensätze durchsucht werden müssen und so im schlimmsten Fall n Zugriffe erforderlich sind, reduziert sich hier die Anzahl der Suchzugriffe erheblich, wie sich der Tabelle 14.2 entnehmen lässt.

Datensätze (n)	Suchzugriffe bei sequenzieller Suche (n)	Suchzugriffe bei Suche mit B-Baum Index $\log_2 n$
10	10	3
100	100	7
1000	1000	10
10000	10000	13
100000	100000	17
1000000	1000000	20

Tabelle 14.2
Ungefähre Anzahl der maximalen Datenzugriffe bei einem Index mit B-Baum

Sie sehen, wie der Index die Anzahl der notwendigen Datenzugriffe mit zunehmender Anzahl der Datensätze drastisch reduziert. Sind bei 10 Datensätzen mit Index noch 30 % der Zugriffe notwendig, so sind es bei 1.000 Datensätzen nur noch 1 %, bei 1.000.000 Datensätzen gar nur noch 0,002 %.

Ein Index kann also Zugriffszeit sparen. Indizes haben aber nicht nur Vorteile, sonst würde man stets alle Spalten und Spaltenkombinationen indizieren. Sie haben auch Nachteile. Diese Nachteile sind im Wesentlichen:

- der zusätzliche Speicherplatz für die Indextabelle
- der zusätzliche Verwaltungsaufwand für den Aufbau und die Pflege des Index

Der B-Baum muss gespeichert und verwaltet werden. Für jeden Datensatz muss ein Eintrag im B-Baum mit allen Werten der indizierten Spalten erfolgen. Das bedeutet, dass bei einem Index der Breite b also b Bytes je Datensatz benötigt werden. Hinzu kommen der Speicher für die Verzweigung des Baums und die generelle Verwaltung. Außerdem vergeben die meisten Systeme den Speicher nur in bestimmten Größen, um eine zu starke Fragmentierung zu vermeiden.

Dies kann schnell dazu führen, dass bei kleinen Tabellen die Indextabelle größer ist als die eigentlichen Daten. Sie sehen es auch an den Indizes für den Primärschlüssel, die standardmäßig bei der Anlage einer Tabelle erzeugt werden. Diese sind bei unseren kleinen Beispieltabellen durchaus schon einmal größer als die Originaltabellen.

Daher lohnt sich die Verwendung von Indizes nur bei größeren Tabellen, als Faustregel sollte die Anzahl der Indexzugriffe also bei $\log_2 n$ unter 1 % liegen. Dies bedeutet, dass man bei Tabellen mit 1.000 Zeilen oder mehr Indizes effektiv einsetzen kann. Allerdings können andere Indexarten oder die Einlagerung von Indextabellen und/oder Datentabellen in den Hauptspeicher auch zu ganz anderen Aussagen führen.

Von großer Bedeutung ist auch die Art der Daten, also die Datentypen. Indizes auf numerischen Feldern stellen zumeist kein besonderes Problem dar.

Soll dagegen nach bestimmten Texten wie beispielsweise Namen oder Produktbezeichnungen gesucht werden, sind im Allgemeinen Felder mit alphanumerischen Werten betroffen. Diese sind mit `CHAR`, `VARCHAR` oder ähnlichen Datentypen definiert und zumeist vergleichsweise groß, beispielsweise 50 Zeichen für einen Namen. Das führt wiederum zu großen Indizes. Bei einigen Datenbanken sind Indizes auf bestimmten textuellen Feldern überhaupt nicht erlaubt. Sind derartige Indizes erlaubt, macht es oft Sinn nicht das gesamte Datenfeld zu verwenden, sondern nur eine bestimmte Anzahl von Zeichen am Wortanfang.

Wird der Index mit

```
CREATE INDEX name ON tbPerson (familienname(10));
```

definiert, hat dies den Vorteil, dass nur die ersten zehn Zeichen des Familiennamens Eingang in den Index finden, dieser somit nur 10 Byte an Reininformation je Datensatz beinhaltet und nicht viel größer als eine numerische Information ist. Andererseits können Sie mit den ersten zehn Zeichen den Namen bereits relativ genau ermitteln und dann den Rest sequenziell erledigen. Die Datenbank kann in vielen Fällen die infrage kommenden Datensätze als linken oder rechten Teilbaum unterhalb des gefundenen Eintrags ermitteln und so unmittelbar weiterarbeiten.

Eine weitere Besonderheit stellen Spalten mit sehr wenigen verschiedenen Werten dar. Im Extremfall sind es nur zwei Werte wie beim Datentyp `Boolean`. Es ist offensichtlich, dass ein Index, der auf einem B-Baum basiert, hier unter Umständen nicht sonderlich effektiv ist – wenn nicht gerade ein Wert selten ist und die Datensätze mit dem seltenen Wert oft gesucht werden.

In solchen Fällen helfen die Bitindizes, die von einigen Datenbanken angeboten werden und die speziell für Spalten mit wenigen Werten entwickelt wurden. Diese Werte werden dann so als Bitmuster verschlüsselt, dass ein schneller Match mit dem entsprechenden Muster bei der Suche möglich ist.

In der Praxis gilt: Probieren geht über Studieren. Wenn Sie also eine Reihe bekannter Zugriffe in Form von `SELECT`-Anweisungen haben, die oft ausgeführt werden müssen und zeitkritisch sind, sollten Sie mit verschiedenen Indizes testen.

Zusammenfassung

Indizes helfen bei:

- der Suche nach bestimmten Datensätzen (`WHERE`-Klausel)
- der Sortierung von Tabellen nach bestimmten Attributwerten (`ORDER BY`-Klausel)
- der Gruppierung nach Attributen (`GROUP BY`- und `HAVING`-Klausel)
- der Abfrage über mehrere Tabellen hinweg (Fremdschlüssel-Primärschlüssel-Beziehungen)

Dafür erfordern Indizes:

- den zusätzlichen Speicherplatz für die Indextabelle und/oder den B-Baum
- den zusätzlichen Verwaltungsaufwand für den Aufbau und die Pflege des Index

Typische Einsatzmöglichkeiten, bei denen der Einsatz eines Index lohnt, sind:

- der häufige direkte Zugriff auf einzelne oder wenige Datensätze über ein oder mehrere bestimmte Datenfelder
- der Zugriff auf Datensätze mit bekanntem Wert oder bekanntem Präfix
- häufig genutzte Fremdschlüsselbeziehungen

Typische Fälle, bei denen ein Index eher zu Nachteilen führt, sind:

- häufige Änderungen im Vergleich zu der Abfragehäufigkeit
- häufige Abfragen von mehr als etwa 30 % der Daten

Natürlich gibt es auch technische Beschränkungen. So gilt in MySQL (ab 5.x) beispielsweise:

- bis zu 64 Indizes pro Tabelle
- bis zu 16 Spalten pro Index
- bis zu 1.000 Byte pro Index

Neben den numerischen Typen dürfen auch `CHAR`, `VARCHAR`, `BLOB` und `TEXT` für Indizes verwendet werden und sie können Präfixe haben, also eine definierte Länge.

14.4 Weitere Anweisungen zur physischen Datenspeicherung

TABLESPACE

Die Daten eines Datenbanksystems werden logisch in Tabellen gespeichert. Physisch werden diese Tabellen auf Festplatten gespeichert. Dazu wird vom Datenbanksystem Speicherplatz reserviert. Die Reservierung dieses Speicherplatzes einschließlich des physischen Speicherortes in Form des Laufwerkes erfolgt normalerweise pro Datenbank. Größere Systeme erlauben aber auch die Einrichtung eigener Tablespaces, also die gezielte Einrichtung von mehreren Speicherbereichen für Tabellen und Indizes. Dann kann bei der Erzeugung von Tabellen und Indizes angegeben werden, in welchem Tablespace diese angelegt werden sollen.

In Oracle existiert dafür eine eigene Anweisung `CREATE TABLESPACE` in anderen Systemen wie MySQL hängt die Verwaltung vom verwendeten Dateiverwaltungssystem ab.

Partitionierung

Neben der Verwaltung von Tablespaces mit der Möglichkeit, innerhalb eines Tablespaces verschiedene Tabellen abzuspeichern, gibt es auch die Möglichkeit, eine Tabelle auf mehrere Tablespaces zu verteilen. Dazu dient die soge-

nannte Partitionierung, bei der eine Tabelle auf mehrere Tablespaces und somit eventuell mehrere physische Speichermedien verteilt werden kann. Die Verteilung kann nach verschiedenen Kriterien erfolgen, im Allgemeinen wird mithilfe bestimmter Wertebereiche des Schlüssels oder anderer Attribute für die Datensätze entschieden, in welcher Partition sie gespeichert werden sollen. Gerade bei sehr großen Tabellen kann eine Partitionierung sinnvoll sein, um beispielsweise in mehreren Partitionen parallel suchen zu können oder Zugriffe im Netz zu vereinfachen.

Weitere Aspekte der physischen Datenhaltung sind die Replikation von Datenbanken, das Laden und Entladen sowie die Sicherung und Wiederherstellung der Datenbestände.

14.5 Prozeduren und Trigger

Die weitere Automatisierung, Konsistenzsicherung und Ergänzung von fehlenden Funktionen in Datenbanken kann mithilfe von Prozeduren und Triggern erfolgen.

Prozeduren

Prozeduren sind letztlich Folgen gespeicherter SQL-Anweisungen, die durch Aufruf der Prozedur immer wieder ausgeführt werden können. Prozeduren können mit eigenen Befehlen erstellt und gelöscht werden, normalerweise mit einem `CREATE PROCEDURE`, hier in der MySQL-Variante. Ein kleines Problem entsteht dadurch, dass die SQL-Anweisungen in der Prozedur selbst wieder mit dem ;-Zeichen beendet werden müssen. Dies begrenzt immer eine SQL-Anweisung (Delimiter). Der Query Browser beendet damit hier die SQL-Anweisung, was nicht gewünscht ist. Daher muss für die Dauer der Definition der Prozedur der Delimiter geändert und am Ende wieder zurückgesetzt werden. Der Query Browser bietet hierfür die Option NEUE PROZEDUR/FUNKTION ANLEGEN, die Sie mit einem rechten Mausklick auf das Schema `kurse` im rechten Fenster öffnen.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS kurse.Personenanzahl $$

CREATE PROCEDURE kurse.Personenanzahl (OUT zahl INT)
BEGIN
   SELECT COUNT(*) INTO zahl FROM tbPerson;
END $$

DELIMITER ;
```

Listing 14.6 Definition einer Prozedur

Der Aufruf kann dann mithilfe einer Folge wie in Listing 14.7 erfolgen, wobei die Anzahl der Datensätze in der Tabelle `tbperson` ermittelt wird.

```
START TRANSACTION;
CALL Personenanzahl(@anzahl);
SELECT @anzahl;
COMMIT;
```

Listing 14.7 Aufruf der Prozedur

Trigger

Trigger dienen dazu, bei der Ausführung von bestimmten Datenbankoperationen wie dem Einfügen, Ändern oder Löschen von Datensätzen zusätzliche Anweisungen ausführen zu können. Typischerweise werden mit Triggern

Konsistenzprüfungen durchgeführt, Datensicherungen vorgenommen oder andere Prozesse gestartet. Das Vorgehen entspricht dem Vorgehen bei Prozeduren.

```
CREATE TRIGGER neuerdozent AFTER INSERT ON tbDozent
   FOR EACH ROW BEGIN
      INSERT tbPerson SET PID = NEW.PID;
END;
```

Listing 14.8
Einfügen eines neuen Dozenten

Trigger werden auf bestimmte Tabellen bezogen und typischerweise gibt es dann zumindest Trigger wie `BEFORE INSERT`, `AFTER INSERT`, `BEFORE UPDATE`, `AFTER UPDATE`, `BEFORE DELETE` und `AFTER DELETE`. Im Trigger können eine oder mehrere SQL-Anweisungen verwendet werden, einschließlich Prozeduren.

Der Trigger wird immer aktiv, wenn die entsprechende Operation in der Datenbank ausgeführt werden soll.

14.6 Application Program Interface

Datenbanken werden zumeist nicht direkt über eine Oberfläche mit einzelnen SQL-Anweisungen angesprochen, sondern sind in vielen Fällen Basis für andere Anwendungen.

Dies erfordert, dass andere Anwendungen auf eine Datenbank zugreifen können. Eine solche Zugriffsmöglichkeit wird von dem Datenbankmanagementsystem über sogenannte API (Application Program Interface) zur Verfügung gestellt.

Ein API für eine relationale Datenbank bietet letztlich immer eine Möglichkeit, SQL-Anweisungen an die Datenbank zu übergeben und das Ergebnis verarbeiten zu können.

Ein API ist stets für eine bestimmte Programmiersprache bestimmt, weil Funktionen in dieser Programmiersprache zur Verfügung gestellt werden müssen. Diese Funktionen sind in der spezifischen Programmiersprache geschrieben.

C-API

In MySQL steht zunächst ein C-API zur Verfügung, mit dem aus C und C++ auf MySQL zugegriffen werden kann. Das C-API besteht aus einer Bibliothek *mysql.h*, die mit einem

```
#include "mysql.h"
```

in ein C-Programm eingebunden wird. Die enthaltenen Funktionen können dann im eigenen C-Programm genutzt werden.

```
mysql_server_init(... Anmeldeinformationen ...);
db = db_connect("kurse");
db_do_query(db, "SELECT * from tbperson");
mysql_close(db);
```

Listing 14.9
Schematische Anmeldung und Ausführung einer SQL-Anweisung

Derartige API-Schnittstellen sind in vielfältiger Form für die meisten Datenbanken und sehr viele Programmiersprachen von C/C++ über C#, Basic-Dialekte, Delphi/Pascal, COBOL bis hin zu den Skriptsprachen wie PHP, Perl oder

Python vorhanden. Nicht jede Sprache lässt sich mit jedem Datenbanksystem verbinden und die Handhabbarkeit der einzelnen API ist unterschiedlich. Manchmal muss auch ein Umweg über eine andere Sprache gegangen werden.

PHP

Es gibt aber auch besonders enge Kopplungen. Das beste Beispiel stellt die Verbindung von PHP und MySQL dar. Mit dem Beispiel in Listing 14.10 wird die Verbindung zu einer MySQL-Datenbank hergestellt, ein Schema ausgewählt und eine SQL-Anweisung ausgeführt.

Dann wird das Ergebnis der Anweisung genutzt, um es als HTML-Tabelle formatiert auszugeben.

Listing 14.10 Zugriff auf MySQL aus PHP

```
$verbindung = mysql_connect("localhost", "root", "masterkey");
if (!verbindung) die ("Der Server ist nicht erreichbar.");
if (!mysql_select_db("kurse", $verbindung))
   die ("Das Schema existiert nicht");
$sql = "SELECT * FROM tbPerson";
$result = mysql_query($sql);
$anzahl_felder = mysql_num_fields($result);
$anzahl_datensaetze = mysql_num_rows($result);
// Jetzt wird die Tabelle zur Ausgabe vorbereitet
echo '<table border="1">';
// Kopfzeile ausgeben
echo "<tr>";
for ($i = 0; $i < $anzahl_felder; $i++) {
   $feldname = mysql_field_name($result, $i);
   echo "<th>$feldname</th>";
}
echo "</tr>";
// Datenzeilen ausgeben
while ($row = mysql_fetch_assoc($result)){
   echo "<tr>";
   foreach ($row as $key => $wert) {
      echo "<td>$wert </td>";
   }
   echo "</tr>";
}
echo "</table>";
```

Zunächst wird eine Verbindung zur Datenbank hergestellt. Dabei werden der Name des Servers, der Name des Benutzers und das Passwort übermittelt. Es wird ein Zeiger `$verbindung` bereitgestellt, der alle relevanten Informationen über die Datenbankverbindung beinhaltet. Anschließend muss ein Schema ausgewählt werden, wobei dieser Zeiger benötigt wird:

```
mysql_select_db("kurse", $verbindung)
```

Ist das Schema ausgewählt, gilt es als Standardschema. Damit können alle folgenden SQL-Anweisungen auf dieses Standardschema zugreifen. Dies wird vorbereitet, indem eine SQL-Anweisung erstellt wird.

```
$sql = "SELECT * FROM tbPerson";
```

Diese SQL-Anweisung wird dann mithilfe des in PHP bereits integrierten Funktionsaufrufs `mysql_query` ausgeführt. In `$result` wird ein Zeiger auf ein assoziatives Feld mit dem kompletten Ergebnis gespeichert.

```
$result = mysql_query($sql);
```

Dann wird die Anzahl der Datenfelder im Ergebnis ermittelt. Diese Anzahl wird dann genutzt, um die Feldnamen als Überschriften ausgeben zu können.

```
for ($i = 0; $i < $anzahl_felder; $i++) {
   $feldname = mysql_field_name($result, $i);
   echo "<th>$feldname</th>";
}
```

Nach der Ausgabe der Feldnamen als Überschrift in einer Tabelle werden die eigentlichen Datensätze ausgegeben. Dazu wird mithilfe der Funktion `mysql_fetch_assoc` jeweils ein Datensatz in die Variable `$row` gelesen. Dieser Datensatz wird hier in einer Schleife in PHP in eine HTML-Tabelle eingefügt.

```
while ($row = mysql_fetch_assoc($result)){
   echo "<tr>";
   foreach ($row as $key => $wert) {
      echo "<td>$wert </td>";
   }
   echo "</tr>";
}
```

Ohne jetzt die Details in PHP im Einzelnen zu klären, ist ersichtlich, wie eng die Integration zwischen PHP und MySQL ist. Durch die direkt in PHP integrierten Funktionen, die alle mit „mysql“ beginnen, ist der Zugriff auf eine Datenbank aus PHP sehr einfach.

14.7 Abschluss

Nach all diesen Seiten bleiben noch viele Themen offen. Über die Speicherung geografischer Daten oder über die Nutzung von Datenbanken für OLAP-Anwendungen konnte nicht gesprochen werden. Die Reparatur von Datenbanken, viele Funktionen der einzelnen Datenbanksysteme, automatische ID-Datenfelder und andere Aspekte konnten nicht so ausführlich betrachtet werden, wie sich das der eine oder andere Leser vielleicht erhofft hätte. Die API hätten sogar mehr als ein eigenes Kapitel verdient. So bleibt stets etwas offen.

Aber dies ist letztlich bei jedem Buch immer wieder der Fall. Ich musste eine Auswahl treffen, Prioritäten setzen und mich immer wieder entscheiden. Dieses Buch soll ein Einstieg in SQL sein und der ist Ihnen hoffentlich gelungen. Ich habe versucht, diesen Aspekt immer im Auge zu behalten.

Nur wer nichts tut, macht keine Fehler. Ich hoffe, dass mir bei der Erstellung dieses Buches nicht zu viele davon unterlaufen sind, und verspreche schon jetzt Besserung.

Wenn Sie Fehler entdecken, Wünsche oder Fragen haben, habe ich dafür die Webseite *`www.serval.de/SQL`* eingerichtet, die auch die E-Mail-Möglichkeit enthält. Ich freue mich auf einen regen Austausch.

A

Anhang: Benutzung der Datenbanksysteme

Wenn Sie sich für eines der Datenbankmanagementsysteme entschieden haben, das Sie für die Beispiele dieses Buches verwenden wollen, aber sich mit dem System noch nicht intensiv beschäftigt haben und nicht genau wissen, wie Sie es handhaben müssen, sind hier einige Hinweise zusammengestellt. Diese Hinweise ersetzen naturgemäß kein Benutzerhandbuch, sollten aber als Starthilfe für einen schnellen Einstieg dienen.

A.1 MySQL

Die Eingabe der SQL-Anweisungen geschieht mithilfe des My SQL Query-Werkzeugs. Dieses können Sie entweder über den MySQL Adminstrator mit EXTRAS/MYSQL QUERY BROWSER oder direkt in der Programmgruppe MYSQL über MYSQL QUERY BROWSER aufrufen (siehe Abbildung 1.1).

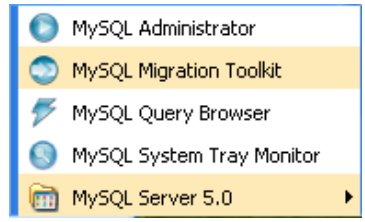


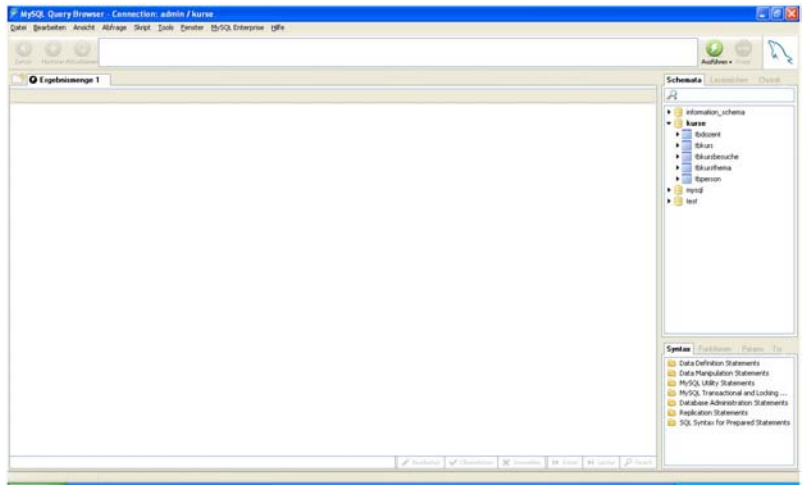
Abbildung 1.1
Aufruf aus Windows

Sie müssen sich danach normal beim MySQL Server anmelden. Der Benutzer *root* ist bei der Erstellung angelegt worden, das Passwort im Standard ist *masterkey*. Die Angabe eines Standardschemas ist keine Pflicht. Sie können bei der folgenden Meldung die Schaltfläche IGNORIEREN wählen. In diesem Fall müssen Sie aber danach im Query Browser im rechten Fenster ein Schema auswählen. Klicken Sie dazu auf *Kurse* (oder falls schon angelegt auf *artikel*). Mit der rechten Maustaste finden Sie die Option zur Auswahl als Standardschema.

Abbildung A.2
Anmeldung beim MySQL Query Browser

Nach der Anmeldung sehen Sie den Standardbildschirm des Query Browsers, wie in Abbildung A.3 dargestellt.

Abbildung A.3
Grundstruktur des Bildschirms des MySQL Query Browsers



Der Bildschirm besteht aus mehreren Teilfenstern, die Sie natürlich verändern können. Die Logik ist aber immer gleich. Sie geben im oberen Fenster eine SQL-Anweisung ein, wie in Abbildung 1.4 zu sehen.

```
SELECT * FROM tbperson;
```

Abbildung 1.4
SELECT-Anweisung im MySQL Query Browser

Zur Ausführung der eingegebenen SQL-Anweisung müssen Sie in jedem Fall die Schaltfläche AUSFÜHREN drücken. Dann wird die eingegebene SQL-Anweisung auf richtige Syntax analysiert und gegebenenfalls ausgeführt. Achten Sie darauf, dass die Anweisung markiert ist. Im oberen Fenster können mehrere SQL-Anweisungen eingegeben werden. Die aktivierte Anweisung erkennen Sie an einem leichten Farbwechsel des Hintergrundes. Im normalen Farbschema ist er bei aktivierter Anweisung weiß.

Anderenfalls erhalten Sie eine Fehlermeldung der Art: „Sie haben versucht, einen leeren String auszuführen. Bitte geben Sie eine SQL-Anweisung in das Bearbeitungsfeld ein und führen sie dann aus.“

Hilfreich ist auch, dass Sie richtige Schlüsselwörter daran erkennen können, dass sie immer sofort farbig hervorgehoben werden. Das Ergebnis der SQL-Anweisung ist in Abbildung 1.5 zu sehen.

Ergebnismenge 1

PID	Familienname	Vorname	PLZ	Strasse	Ort	Geburtsdatum
1	Weiss	Peter	30529	Palmstraße 6	Hannover	1963-11-07
2	Bucz	Susanne	30531	Heinestraße 23	Hannover	1976-04-06
4	Karmann	Thomas	29227	Trift 28	Celle	1954-08-04
5	Klötzer	Karl	29221	Bahnhofstraße 2	Celle	1971-03-13
6	Weiss	Karin	30529	Palmstraße 6	Hannover	1962-10-05
7	Weiss	Peter	38134	Glanweg 4	Braunschweig	1974-03-02
8	Meier	Kathrin	38154	Welfenallee 23	Braunschweig	1981-05-03
9	Schmidt	Karl	30529	Lavesallee 23	Hannover	1949-06-25
10	Müller	Claudia	29596	In den Fuhren 12	Breitenhees	NULL
11	Lisewski	Bernd	30890	Roggenkamp 10	Barsinghausen	1960-06-06
15	Martens	Melanie	29221	Horstweg 258	Celle	1911-02-17
17	Schlachter	Dieter	29227	Mondhagen 43	Celle	1961-02-02
23	Peredy	Helmut	29221	Mauernstraße 2	Celle	1956-02-23
24	Ruppert	Nicola	29301	Welfenallee 23	Bergen	1962-02-25
25	Sander	NULL	29223	Marxallee 12	Celle	1953-02-05
26	Cromberg	Jörg	38152	Nordring 13	Braunschweig	1991-06-07
31	Schulze	Tanja	29308	Berliner Ring 23	Winsen	1992-11-09
32	Winter	Petra	29320	Immenhoop 51	Hermannsburg	1989-12-30
34	Plate	Ulrich	30529	Gutenberggasse 5	Hannover	1986-12-02
37	Magerkurth	Melissa	29336	Am Tümpel 3	Nienhagen	1951-09-04

Abbildung 1.5
Ergebnis der Abfrage SELECT * FROM tbPerson;

Die Anzahl der Datensätze sowie die Dauer der Abfrage können Sie am unteren Ende des Bildschirms sehen (siehe Abbildung 1.6). Die Abfragedauer besteht dabei nicht nur aus der Dauer der Datenbankabfrage, sondern auch aus allen Verarbeitungs- und Aufbereitungszeiten.

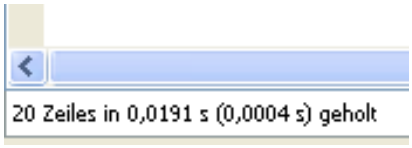
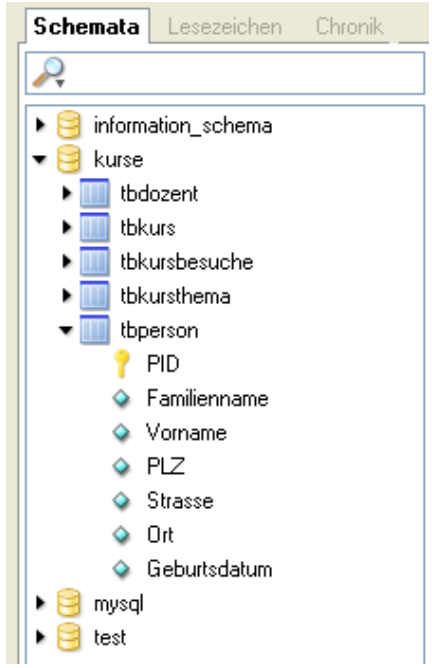


Abbildung 1.6
Anzahl der Ergebnisdatensätze, Abfragezeit (reine Datenbankabfragezeit)

In diesem Buch sind viele Beispiele für SQL-Anweisungen enthalten. Diese können Sie nutzen, um die Oberfläche zu testen. In manchen Fällen müssen Sie die Anweisung vielleicht etwas anpassen oder wollen sie erweitern, um etwas auszuprobieren. Um eine sinnvolle SQL-Anweisung eingeben zu können, können Sie unmittelbar auf die beiden Fenster am rechten Bildschirmrand zurückgreifen. Dort finden Sie zum einen eine Übersicht über das Schema mit den Tabellen und innerhalb der Tabellen wiederum über die darin enthaltenen Datenfelder. Durch Klicken auf die kleinen Pfeile vor dem Schema und den Tabellen können Sie die Detailansichten jeweils ein- oder ausschalten. Im Prinzip haben Sie die dreistufige Hierarchie wie in Abbildung A.7 angegeben zur Verfügung.

Abbildung A.7
Schema kurse mit fünf Tabellen



Standardschema

Das Schema *kurse* haben Sie bereits am Anfang erzeugt. Wenn Sie sich mit einem Standardschema angemeldet haben, wird dieses hier bereits geöffnet. Wenn Sie sich ohne Standardschema angemeldet haben, müssen Sie dieses jetzt hier öffnen. Dies ist wichtig, damit MySQL weiß, aus welchem Schema die Tabellen und Datenfelder stammen, die Sie in der SQL-Anweisung verwenden. Klicken Sie mit der rechten Maustaste auf ein Schema, können Sie dies zum Standardschema machen. Dies ist wichtig, damit die entsprechenden Tabellen ohne weitere Qualifizierung gefunden werden können.

Wenn Sie übrigens den Inhalt einer Tabelle nur einfach einmal schnell testen wollen, können Sie mit einem Doppelklick auf einen Tabellennamen eine entsprechende SELECT-Anweisung erzeugen und mit einem weiteren Doppelklick oder mit der Schaltfläche AUSFÜHREN den Inhalt anzeigen.

Oberhalb des Fensters finden Sie noch ein kleines Suchfenster zum Suchen innerhalb des Schemafensters, wenn Sie beispielsweise einmal ein Feld suchen und überhaupt nicht mehr wissen, wo sich dieses versteckt.

Wenn Sie Hilfe insbesondere für komplexere Anweisungen und Funktionen benötigen, steht Ihnen schließlich unten rechts noch ein Fenster zur Verfügung.

Sie können mehrere Tabs öffnen, um mehrere Anweisungen parallel verwenden zu können. Sie können Ihre Anweisungen als Query speichern und später wieder laden und ausführen.

A.2 Oracle

Um die SQL-Anweisungen in Oracle einzugeben, steht neben den zeilenorientierten Oberflächen wie SQLplus oder PL/SQL insbesondere die browserorientierte Oberfläche zur Verfügung. Hierzu öffnen Sie in Windows über START die Programmgruppe ORACLE DATABASE 10G EXPRESS EDITION. Sie finden die in Abbildung 1.8 angegebenen Elemente.

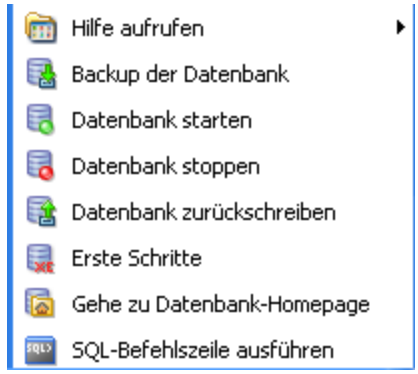
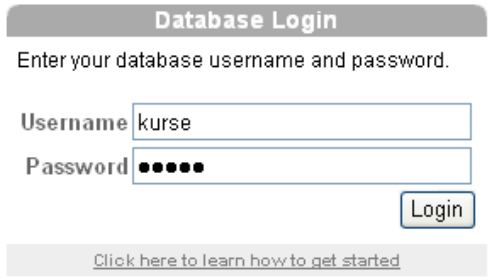


Abbildung 1.8
Programmelemente von Oracle Database 10g Express Edition

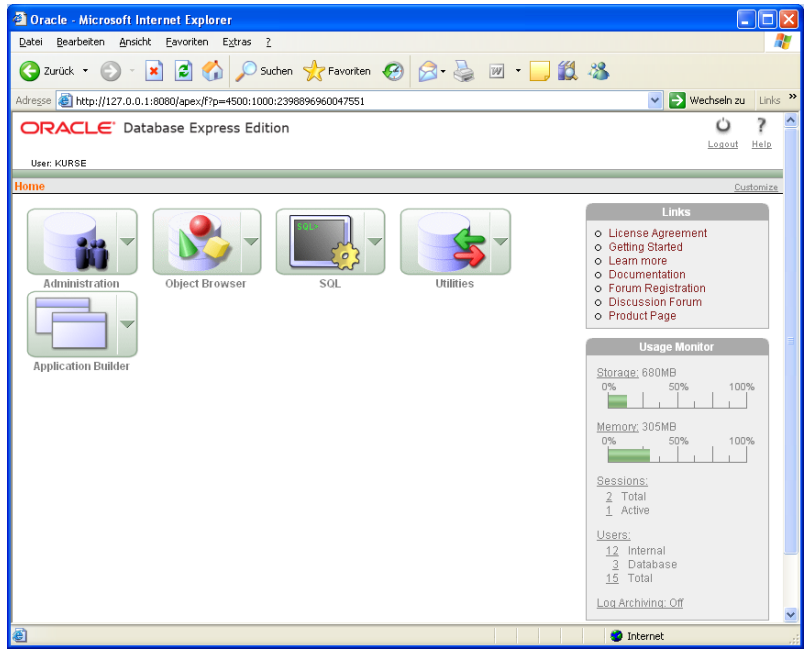
Interessant ist jetzt der Punkt GEHE ZU DATENBANK-HOMEPAGE, mit dem Sie Ihren Browser mit der lokalen Startseite Ihrer Oracle-Datenbank starten. In der Startseite finden Sie die Möglichkeit, sich bei Ihrer Datenbank anzumelden. Dafür stehen sowohl die Superuser *SYS* und *SYSTEM* als auch der bereits im Rahmen der Einrichtung der Beispieldatenbanken erzeugte Benutzer *kurse* zur Verfügung. Geben Sie den Benutzer zusammen mit dem von Ihnen festgelegten Passwort ein und melden Sie sich an.

Abbildung A.9
Anmeldung im Browser in der lokalen Datenbank



Nach der Anmeldung gelangen Sie in das Übersichtsfenster der Oberfläche (siehe Abbildung A.10).

Abbildung A.10
Übersichtsfenster Oracle



Hier finden Sie die zentralen Anwendungen:

- ADMINISTRATION: Dient der Verwaltung des Datenbankzugriffs über Benutzer sowie des physikalischen Speichers und der Netzwerkeinstellungen. Diesen Bereich müssen Sie zunächst nur beachten, wenn Sie die Passwörter beispielsweise für die Superuser ändern wollen. Ratsam ist dies, wenn Sie sich im Netz befinden und Ihre Inhalte schützen wollen.
- OBJECT BROWSER: Der Object Browser dient der Übersicht über die Datenbank. Mit ihm können Sie insbesondere die Struktur Ihrer Tabellen ermitteln. Es ist of ratsam, im Browser ein Fenster oder eine Registerkarte mit diesen Inhalten geöffnet zu halten, um leichter nachschlagen zu können.

- SQL: Hier erfolgt die eigentliche Eingabe und Verwaltung der SQL-Anweisungen. Ein Fenster mit diesem Werkzeug sollten Sie stets geöffnet halten, um die SQL-Anweisungen testen zu können.
- UTILITIES: Dies stellt die Oberfläche einer Reihe von Zusatzprogrammen dar, die Oracle anbietet. Obwohl viele davon den Umgang mit der Datenbank deutlich erleichtern, sollten sie hier nicht den Schwerpunkt bilden, da es um SQL geht.
- APPLICATION BUILDER: Weiter gehende Anwendungen auf der Datenbank.

Sie sollten testweise einmal SQL aufrufen, wie in Abbildung 1.11 dargestellt.

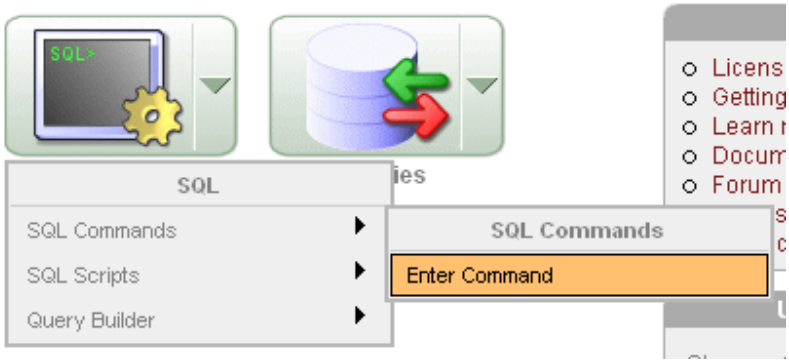


Abbildung 1.11
Aufruf der SQL-Eingabeoberfläche

Sie können in dem jetzt angebotenen Fenster die komplette Eingabe, Ausführung und Ergebniskontrolle der SQL-Anweisungen ausführen (siehe Abbildung 1.12).

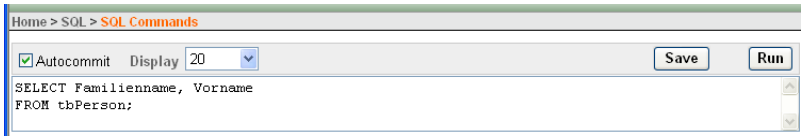


Abbildung 1.12
SQL-Anweisung in Oracle

Der obere Teil des Fensters erlaubt die direkte Eingabe von SQL-Anweisungen. Geben Sie testweise das angezeigte Beispiel an. Zum Aufruf des Befehls müssen Sie die Schaltfläche RUN anklicken.

Das Ergebnis sollte etwa der in Abbildung A.13 dargestellten Tabelle entsprechen. Beachten Sie, dass die angezeigte Zeilenanzahl von der in DISPLAY angegebenen Maximalzahl abhängt. Gegebenenfalls müssen Sie diese erhöhen oder blättern.

Abbildung A.13
Ergebnis der SQL-Anweisung

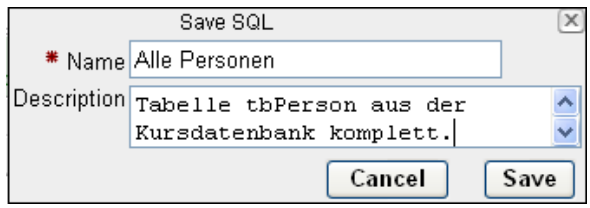
Results Explain Describe Saved SQL History

FAMILIENNAME	VORNAME
Weiss	Peter
Bucz	Susanne
Karmann	Thomas
Kloetzer	Karl
Weiss	Karin
Weiss	Peter
Meier	Kathrin
Schmidt	Karl
Mueller	Claudia
Lisewski	Bernd
Peredy	Helmut
Schlachter	Dieter
Martens	Melanie
Ruppert	Nicola
Sander	-
Cromberg	Joerg
Schulze	Tanja
Magerkurth	Melissa
Winter	Petra
Plate	Ulrich

20 rows returned in 0,00 seconds CSV Export

Sie können einmal eingegebene SQL-Anweisungen speichern. Verwenden Sie dazu die Schaltfläche SAVE. Damit speichern Sie die eingegebene SQL-Anweisung ohne Daten. Oracle fragt Sie nach einem Namen und einer Beschreibung, also nach Metadaten zu der erstellten Anweisung (siehe Abbildung A.14). Geben Sie eine Information ein, die Ihnen später hilft, die Anweisung wiederzufinden. Im Fall der Buchbeispiele kann dies eine inhaltliche Angabe wie im Beispiel oder einfach auch die Seitennummer sein.

Abbildung A.14
Metadaten zu einer SQL-Anweisung



Sie können jederzeit auf einmal gespeicherte Abfragen zurückgreifen, indem Sie im Übersichtsfenster die Option SAVED SQL wählen. Wie in Abbildung 1.15 dargestellt können Sie aus allen gespeicherten Abfragen wählen. Über den OWNER können Sie das Schema einschränken, beispielsweise auf die Beispieldatenbank *KURSE*.

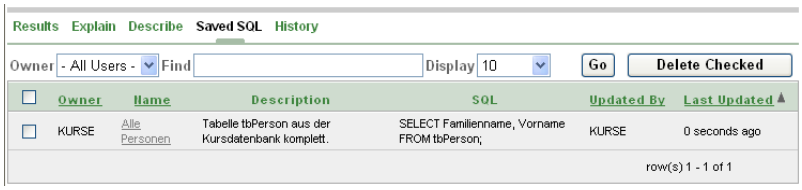

Results Explain Describe Saved SQL History
Owner - All Users - Find Display 10 Go Delete Checked
Owner Name Description SQL Updated By Last Updated
KURSE Alle Personen Tabelle tbPerson aus der Kursdatenbank komplett. SELECT Familienname, Vorname FROM tbPerson; KURSE 0 seconds ago
row(s) 1 - 1 of 1

Abbildung 1.15
Übersicht über die gespeicherten SQL-Anweisungen

Interessant ist auch noch die Option DESCRIBE im Übersichtsfenster. Nachdem Sie die SQL-Anweisung `DESCRIBE` mit dem Namen einer Tabelle im oberen Fenster ausgeführt haben, können Sie sich im Übersichtsfenster den Aufbau der kompletten Tabelle ansehen und dann bei der Erstellung der eigentlichen SQL-Anweisung jeweils unten auf den Namen der Tabelle und der Spalten (Columns) klicken, die Sie benötigen. Dies erspart das Tippen der Namen und verhindert Schreibfehler.

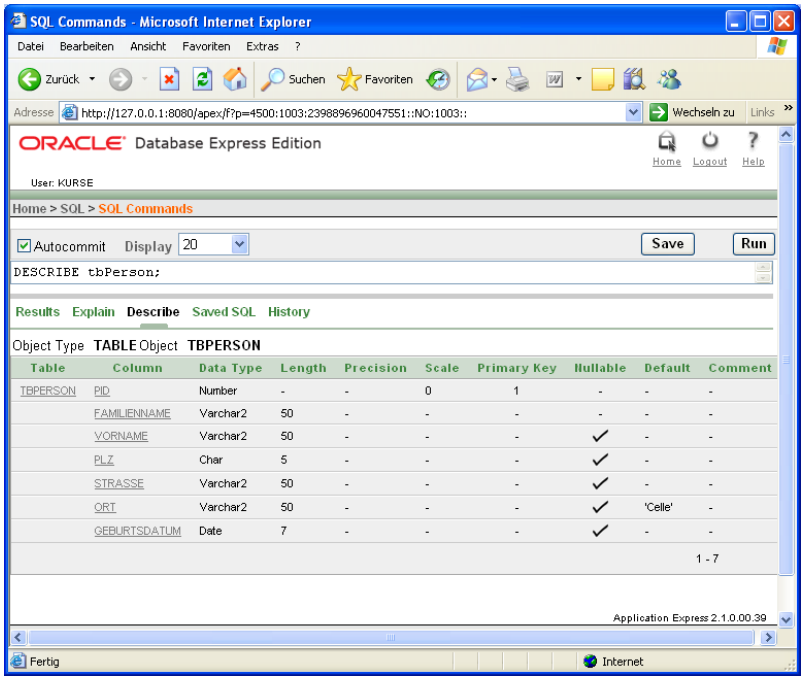

SQL Commands - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://127.0.0.1:8080/apex/f?p=4500:1003:2398896960047551::NO:1003::
Wechseln zu Links
ORACLE Database Express Edition
Home Logout Help
User: KURSE
Home > SQL > SQL Commands
Autocommit Display 20 Save Run
DESCRIBE tbPerson;
Results Explain Describe Saved SQL History
Object Type TABLE Object TBPERSON
Table Column Data Type Length Precision Scale Primary Key Nullable Default Comment
TBPERSON PID Number - - 0 1 - - -
FAMILIENNAME Varchar2 50 - - - - - -
VORNAME Varchar2 50 - - - ✓ - -
PLZ Char 5 - - - ✓ - -
STRASSE Varchar2 50 - - - ✓ - -
ORT Varchar2 50 - - - ✓ 'Celle' -
GEBURTSDATUM Date 7 - - - ✓ - -
1 - 7
Application Express 2.1.0.00.39
Fertig
Internet

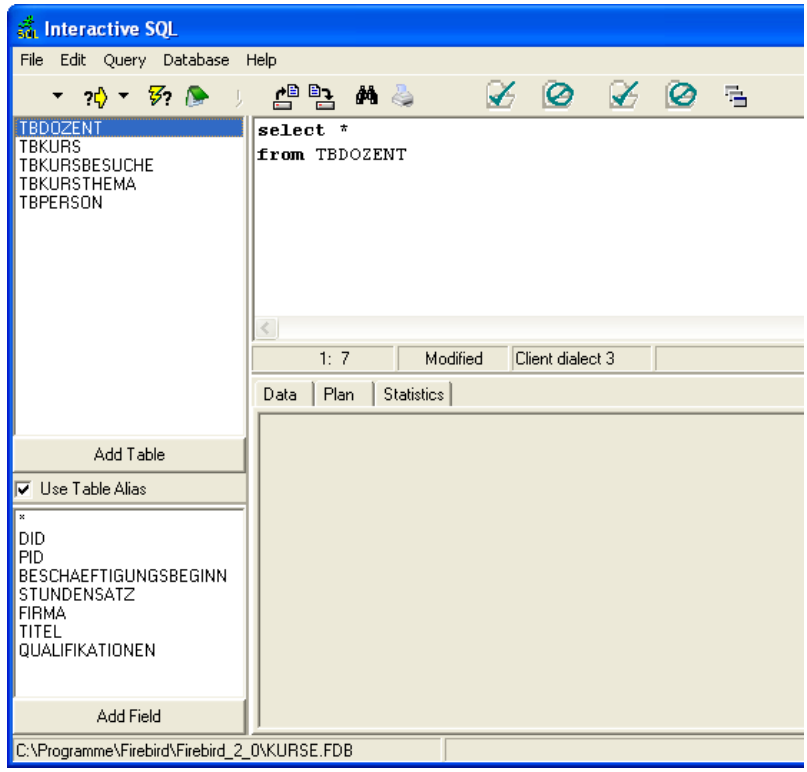
Abbildung 1.16
Beschreibung einer Tabelle mit DESCRIBE

A.3 Firebird

Firebird ist als Datenbank konzipiert, die hauptsächlich in andere Anwendungsprogramme integriert wird. Die hier verwendete Oberfläche ist also eher untypisch für Firebird. Firebird ist aus dem System *Interbase* entstanden, das ursprünglich die Firma *Borland*, die später eine Zeitlang *Inprise* hieß, entwickelt hat. Heute werden diese Systeme alle von *CodeGear* betreut, dass zu *Embarcadero* gehört. Entsprechende Informationen finden Sie dort. Im Internet bietet *www.firebirdsql.org* Informationen zu dem freien Firebird. In den meisten Fällen können Sie auch gut auf die Interbase-Dokumentationen zurückgreifen.

Die Abbildung A.17 zeigt die Eingabeoberfläche, die Sie aus dem *IBOConsole*-Programm heraus mit dem Menübefehl TOOLS/INTERACTIVE SQL starten können. Hier können Sie Ihre SQL-Anweisungen direkt eingeben. Als Hilfe können Sie die beiden Fenster im linken Bereich verwenden. Das obere Fenster zeigt alle Tabellen des aktuellen Schemas. Sie können durch einen einfachen Klick auf einen Tabellennamen die Felder der Tabelle unten anzeigen lassen. Durch einen Doppelklick im unteren Bereich können Sie Feldnamen direkt in Ihre SQL-Anweisung übernehmen. Durch einen Doppelklick im oberen Bereich können Sie den Tabellennamen beispielsweise für die `FROM`-Klausel übernehmen.

Abbildung A.17
Eingabe von SQL-Anweisungen



Die Ausführung der Anweisung erfolgt grundsätzlich mit dem „Blitzsymbol“. Das Ergebnis der Anweisung wird dann im unteren Bereich angezeigt oder es wird eine entsprechende Fehlermeldung angezeigt.

Das Ergebnis der Abfrage kann als Tabelle gespeichert und dann mit anderen Werkzeugen weiterverarbeitet werden.

Aber auch die SQL-Anweisung kann gespeichert werden. Dabei werden grundsätzlich alle ausgeführten Anweisungen gepuffert und können wieder zurückgeholt werden. Stehen weitere Abfragen zur Verfügung, kann auch wieder nach vorn geblättert werden.

Neben dieser automatischen Speicherung können SQL-Anweisungen auch als SQL gespeichert werden und anschließend wieder als SQL-Script geladen werden.

Es steht über die Menüoption HELP eine komplette SQL-Hilfe zur Verfügung. Die weiteren Optionen dienen weitgehend der Einbindung der SQL-Anweisungen beziehungsweise sind hier zu speziell.

Neben dem Aufruf der SQL-Eingabe über TOOLS bietet IBOConsole eine Reihe von Informationsmöglichkeiten über eine Datenbank, die sich durch eine Aktivierung der Datenbank, hier KURSE.FDB sichtbar machen lassen. Interessant sind dabei zunächst vor allem die TABLES, die sich durch einen Klick im linken Bereich wie alle anderen Elemente auch sichtbar machen lassen. Auch die in diesem Buch angelegten Domänen, Views und weitere Elemente können hier sichtbar gemacht werden.

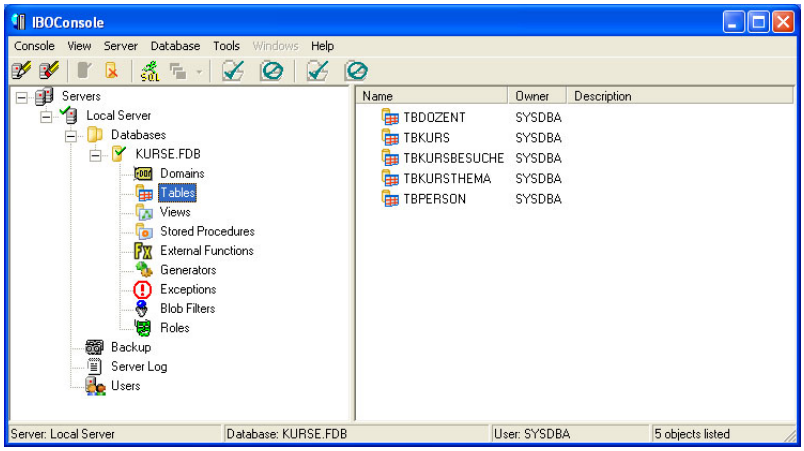
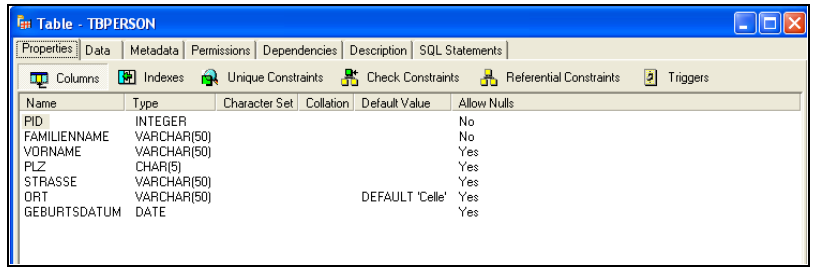


Abbildung 1.18
Übersicht Datenbank

Ein Doppelklick auf eine Tabelle, beispielsweise TBPERSON stellt eine ganze Reihe an Informationen zur Verfügung. So lassen sich die Attribute mit allen Eigenschaften (PROPERTIES) und der Inhalt der Tabelle (DATA) darstellen (auf der DATA-Seite müssen Sie die Tabelle noch mit OPEN öffnen). Aber auch die SQL-DDL zur Erzeugung der Tabelle (METADATA) sowie die Zugriffsrechte (PERMISSIONS) auf die Tabelle lassen sich anzeigen. Fremdschlüsselbeziehungen (DEPENDENCIES), eine Beschreibung für den Benutzer (DESCRIPTION) sowie

vorgenerierte SQL-Anweisungen (SQL STATEMENTS) runden die Informationen ab. Gerade wenn es in diesem Buch um die DDL und die DCL geht, kann diese Sicht genutzt werden, um die Auswirkungen der einzelnen Anweisungen zu prüfen.

Abbildung A.19
Eigenschaften eine Tabelle



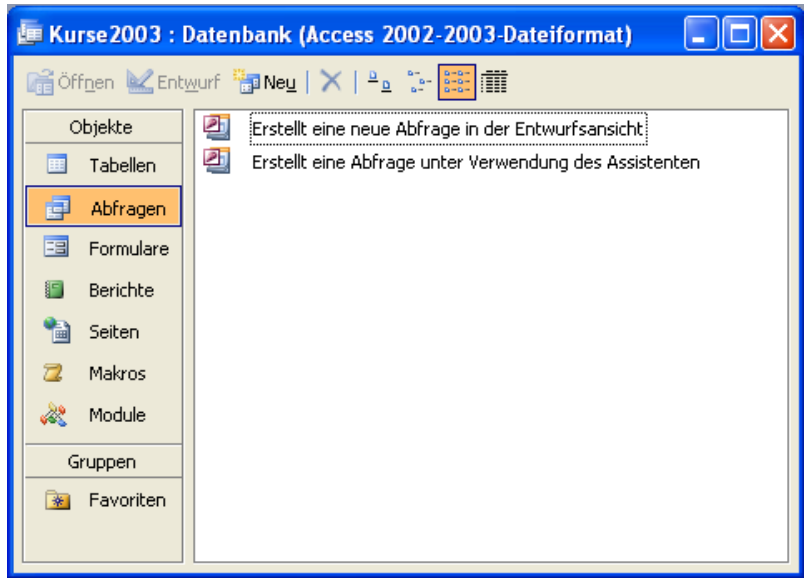
A.4 MS Access

1.4.1 SQL-Anweisungen eingeben

MS Access bietet neben der reinen SQL-Eingabemöglichkeit eine grafische Oberfläche für die Erstellung einer Abfrage. Dabei wird die Abfrage in einer Oberfläche zusammengestellt. Anschließend generiert MS Access aus diesen Angaben eine SQL-Anweisung. Diese SQL-Anweisung kann dann wie bei allen anderen Datenbanksystemen ausgeführt werden und liefert die Ergebnisdatensätze.

Dieses Vorgehen hat den Vorteil, dass der Anwender nicht unbedingt SQL beherrschen muss, um eine SQL-Anweisung zu erstellen. Dies macht MS Access attraktiv für kleine Datenbankanwendungen, die auch in Fachabteilungen oder im privaten Umfeld nutzbar sind.

Abbildung A.20
Übersicht über vorhandene Abfragen



Abfragen sind in MS Access eigene Datenbankobjekte. Im Datenbankfenster sind nach den Tabellen die übrigen Datenbankobjekte aufgelistet. Abfragen, Formulare, Berichte, Seiten, Makros, Module sowie die Gruppen sind im Gegensatz zu den Tabellen Objekte, die nicht selbst Daten speichern, sondern auf den Daten in den Tabellen beruhen, diese also anzeigen, kombinieren und später auch ändern und löschen.

Abfragen bilden dabei die wichtigsten Objekte. Wenn Sie im Datenbankfenster links die ABFRAGEN auswählen, wird rechts die komplette Liste dargestellt.

Sie können jetzt eine neue SQL-Anweisung „eingeben“, indem Sie eine neue Abfrage erstellen. Dazu können Sie im Datenbankfenster entweder die Schaltfläche NEU wählen oder einen der Assistenten aufrufen.

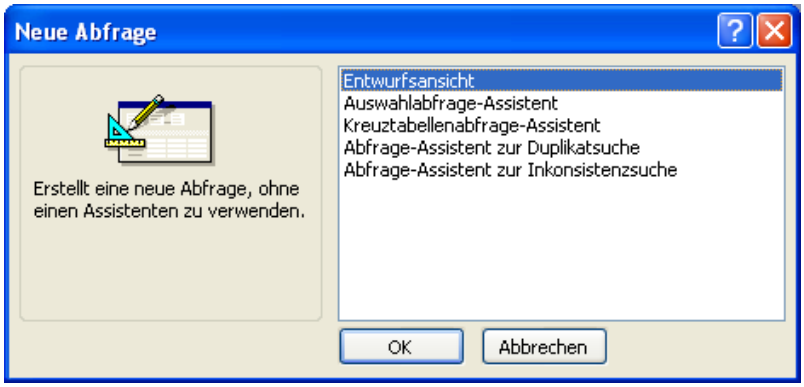


Abbildung 1.21
Auswahl für die Erstellung einer neuen Abfrage

Die Assistenten sollen uns hier nicht so sehr interessieren, da wir ohnehin mehr am puren SQL interessiert sind. Daher sollte der Einstieg im Rahmen dieses Buches stets über die Schaltfläche NEU mit der Entwurfsansicht erfolgen. Sie gelangen zu einem Dialog wie in Abbildung 1.21. Wählen Sie die ENTWURFSANSICHT. Sie kommen jetzt in Abbildung 1.22.

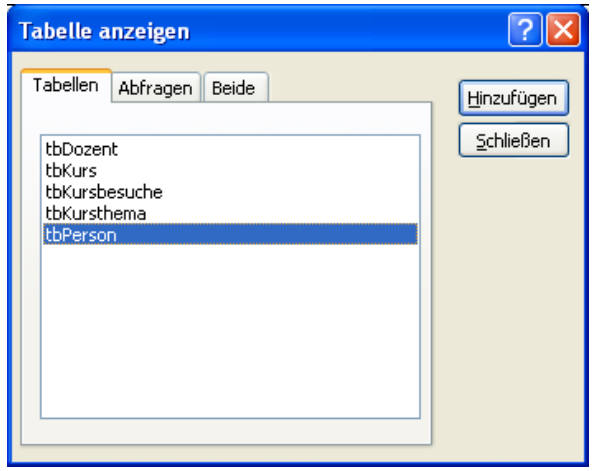


Abbildung 1.22
Auswahl der Datenquellen für die Abfrage

Hier wählen Sie die Datenquelle für Ihre Abfrage. Datenquellen sind zunächst die Tabellen der Datenbank. Sie legen also fest, aus welcher Tabelle (oder aus welchen Tabellen) Daten ausgewählt werden sollen.

Sie sehen übrigens, dass Sie Daten nicht nur aus Tabellen, sondern auch aus anderen Abfragen gewinnen können. Dies sind aber keine originären Daten, sondern Daten, die die anderen Abfragen ihrerseits aus Tabellen (oder wiederum aus Abfragen) ermitteln.

Im Normalfall werden hier Tabellen gewählt. Wählen Sie also die Tabellen aus, die Sie als Basis verwenden wollen. Die Tabellen werden in die sogenannte Entwurfsansicht übernommen, die Sie in Abbildung A.23 sehen. Dies ist das zentrale Fenster für die Erstellung der SQL-Anweisung, die sogenannte grafische Oberfläche. In dieser Entwurfsansicht wird, so weit möglich, die gesamte Abfrage erstellt.

Wesentlich sind zunächst die ausgewählten Tabellen im oberen Teil des Fensters. Innerhalb der Tabellen sind die Datenfelder der Tabellen aufgelistet. Sie können jetzt die Felder auswählen, die von der SQL-Anweisung gezeigt werden sollen. Wählen Sie die Felder entweder per Doppelklick auf den Feldnamen oder durch Ziehen mit der Maus in den unteren tabellenartigen Teil aus. Im Beispiel sind die `PID`, der `Familienname` und der `Vorname` ausgewählt.

Wenn Sie Felder ausgewählt haben, die Sie nicht benötigen, löschen Sie das Feld in der unteren Tabelle. Markieren Sie dazu die gesamte Spalte oder löschen Sie den kompletten Feldnamen aus der Tabelle.

Wichtig für die spätere Nutzung sind die Zeilen SORTIERUNG und KRITERIEN. Die Zeile SORTIERUNG erlaubt einen Eintrag, der die Ergebnisliste entsprechend den Werten eines Feldes sortiert. Bei den KRITERIEN können Einschränkungen der Datensätze eingetragen werden. Die Sortierungen werden von links nach rechts angewendet. Gibt es mehrere Sortierungen, müssen Sie also gegebenenfalls die Spalten tauschen, um die gewünschte Sortierreihenfolge herzustellen.

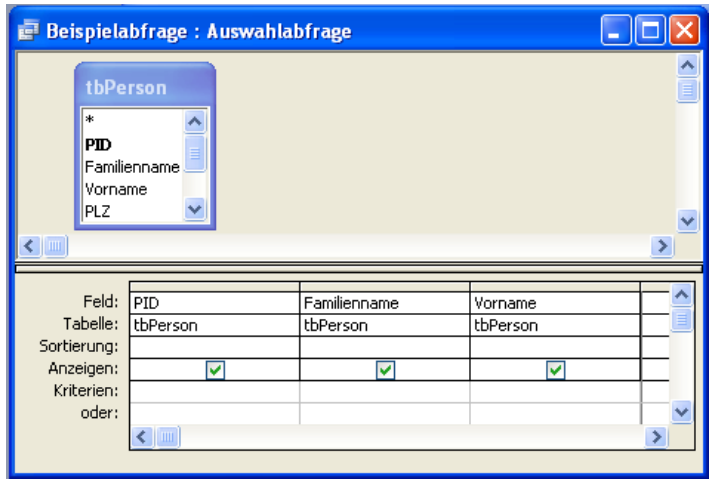


Abbildung A.23
Entwurfsansicht = Fenster für die Zusammenstellung der SQL-Anweisung

Die Entwurfsansicht ist nur eine von verschiedenen Sichten auf dieselbe Information. Die drei wesentlichen Sichten sind neben der Entwurfsansicht die Datenblattansicht und die SQL-Ansicht. Die Umschaltung zwischen den verschiedenen Sichten erfolgt mit dem Symbol oben links, das in der Entwurfsansicht ein tabellenähnliches Bild enthält. Wenn Sie auf den Pfeil rechts neben dem Symbol klicken, werden Ihnen die verschiedenen Ansichten zur Auswahl angeboten (siehe Abbildung 1.24).

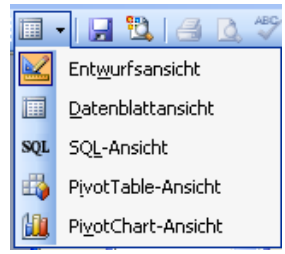


Abbildung 1.24
Die verschiedenen Ansichten einer Abfrage

Wählen Sie die SQL-Ansicht, so sehen Sie die SQL-Anweisung, die der in der Entwurfsansicht getroffenen Auswahl entspricht (siehe Abbildung 1.25).

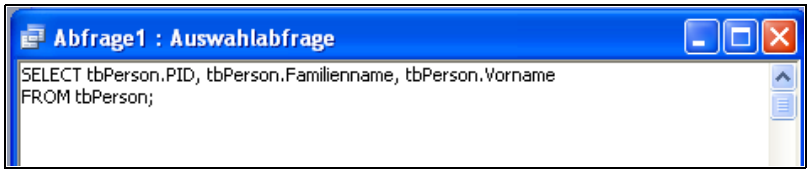


Abbildung 1.25
SQL-Anweisung entsprechend der Auswahl in der Entwurfsansicht

Sie können also in MS Access unmittelbar sehen, welche SQL-Anweisung Sie über die grafische Oberfläche erzeugt haben. Das Ganze funktioniert (mit einigen Einschränkungen) übrigens auch umgekehrt. Sie haben also bei den Beispielen dieses Buches oft die Wahl, ob Sie die grafische Oberfläche nutzen oder ob Sie die Anweisung direkt in die SQL-Ansicht eingeben. In diesem Fall haben Sie eine vergleichbare Situation wie bei den anderen Systemen.

Durch Wahl der Datenblattansicht gelangen Sie zu den eigentlichen Daten. Das Ergebnis der SQL-Abfrage wird als Datensatztabelle dargestellt (siehe Abbildung A.26). Sie können jetzt wieder zur Entwurfsansicht oder zur SQL-Ansicht zurückkehren und beliebig zwischen den Ansichten wechseln.

Schließlich können Sie die Abfrage speichern, indem Sie sie einfach schließen. Das ist etwas ungewohnt, entspricht aber dem Prinzip von MS Access, alle Informationen beim Schließen eines Fensters zu speichern. Sie werden aber nach einem Namen für die Abfrage gefragt (siehe Abbildung A.27). Beachten Sie, dass bei der Speicherabfrage nur die Struktur der Abfrage, also die SQL-Anweisung, gespeichert wird, nicht die Daten. Diese bleiben unverändert in den Tabellen.

Abbildung A.26
Die Datenblattansicht ist das Ergebnis der Abfrage.

Abbildung A.27
Speichern der Abfrage

Danach erscheint die Abfrage in der Liste der verfügbaren Abfragen im Datenbankfenster, wie in Abbildung 1.28 dargestellt.

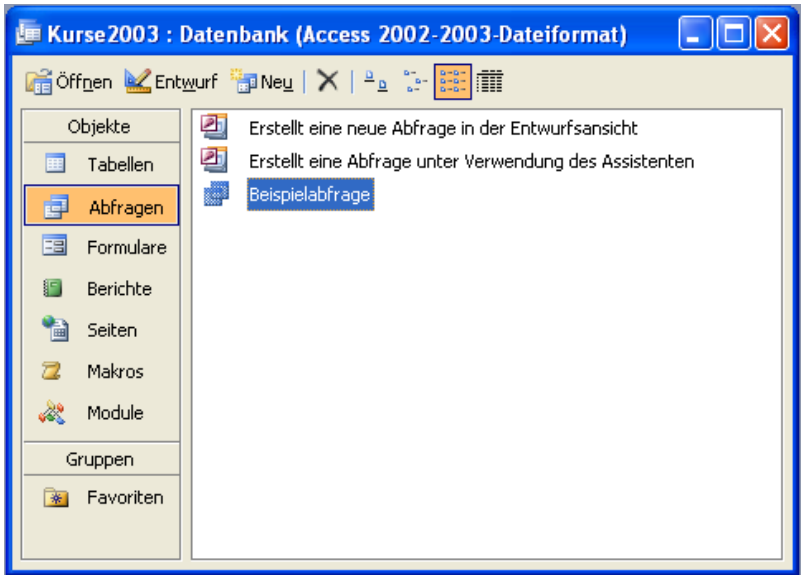


Abbildung 1.28
Übersicht nach Erstellung der Abfrage

A.4.2 Die Daten aus Excel importieren

Datenimport in MS Access

Wir wollen die Gelegenheit nutzen, beispielhaft auch eine neue Datenbank anzulegen. Damit haben Sie nicht nur eine Datenbank zur Verfügung, die Sie einerseits kopieren und andererseits immer wieder neu aufbauen und erweitern können, wenn Sie die Übungsdatenbank aus irgendwelchen Gründen zerstört haben, sondern auch ein Verfahren, eigene Dateien beispielsweise aus Excel in eine Datenbank zu übernehmen.

Wählen Sie die NEU und dann die Option LEERE DATENBANK und bestätigen Sie diese mit OK.

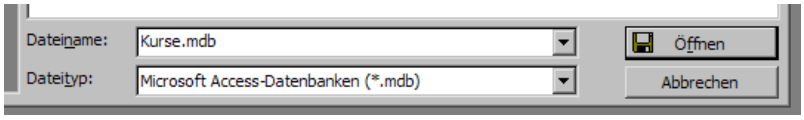
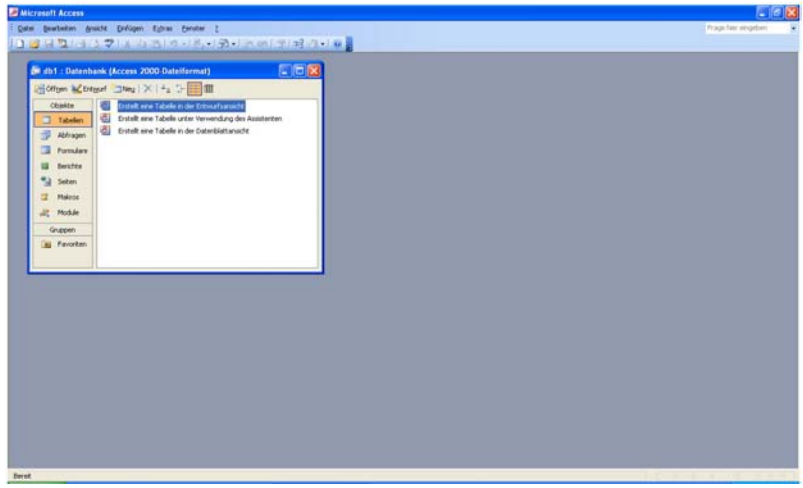


Abbildung 1.29
Anlage einer MS Access-Datenbank.

Sie können jetzt die komplette Datenbank anlegen, indem Sie in einem eigenen Verzeichnis einen Namen – hier Kurse – wählen und die Auswahl mit ÖFFNEN bestätigen. Sie erhalten das sogenannte Datenbankfenster.

Sie sehen, dass links die Option TABELLEN aktiviert ist. Rechts werden nur einige Assistenten angezeigt, aber keine Tabellen. Sie haben also jetzt eine „leere“ Datenbank erzeugt. Diese Datenbank könnten Sie jetzt manuell mithilfe der Assistenten anlegen. Tatsächlich geschieht dies wieder mit SQL, wie wir es bereits von MySQL kennen. MS Access versteckt die SQL-Anweisungen aber noch einmal besonders und erlaubt es so, sehr einfach Tabellen mit Inhalt über die Oberfläche zu importieren.

Abbildung A.30
Datenbankfenster von MS Access mit leerer Datenbank



Die Schritte zum Import einer Tabelle in MS Access sind folgende:

1. Prüfen Sie, ob Sie in der linken Optionsleiste die Auswahl TABELLEN getroffen haben. Das Datenbankfenster sieht für alle Auswahlen fast identisch aus, führt aber natürlich zu sehr unterschiedlichen Ergebnissen, je nachdem, ob Sie mit Tabellen, Abfragen oder beispielsweise Makros arbeiten.
2. Wählen Sie im Datenbankfenster die Schaltfläche NEU. Sie erhalten ein neues kleines Fenster NEUE TABELLE.

Abbildung A.31
Importoption für die Erstellung neuer Tabellen aus vorhandenen Dateien

3. Wählen Sie hier die Option TABELLE IMPORTIEREN und bestätigen Sie die Auswahl mit OK.
4. Sie sehen ein Fenster zur Dateiauswahl, aus dem Sie die Quelldatei für die anzulegende Tabelle auswählen können. Wenn Sie die CD dieses Buches noch nicht verwendet haben, legen Sie sie ein und wählen darauf die Ordner *Datenbanken/MSAccess/Kurse*. Achten Sie darauf, dass Sie unten im Auswahlfenster den Typ, der zu importierenden Dateien, auf Excel umstellen (siehe Abbildung A.32). Voreingestellt ist der Typ für MS Access-Datenbanken.

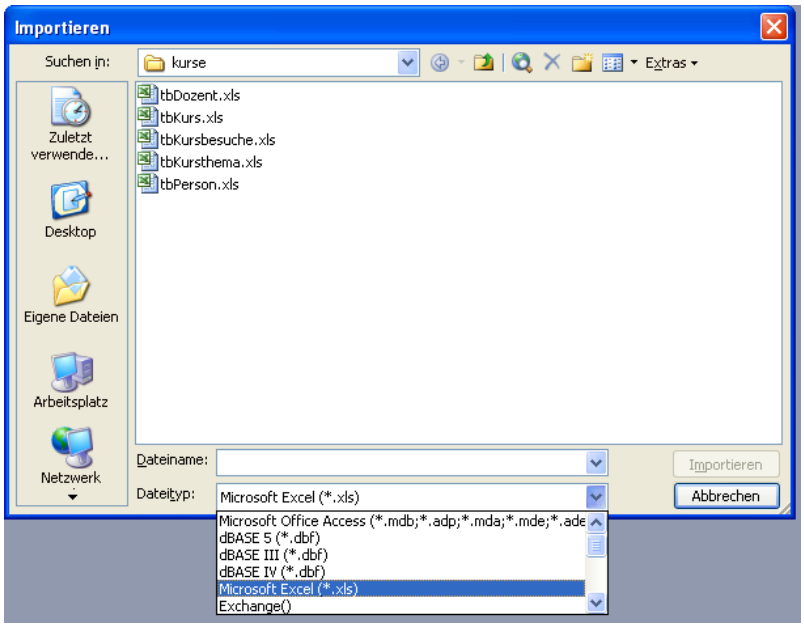
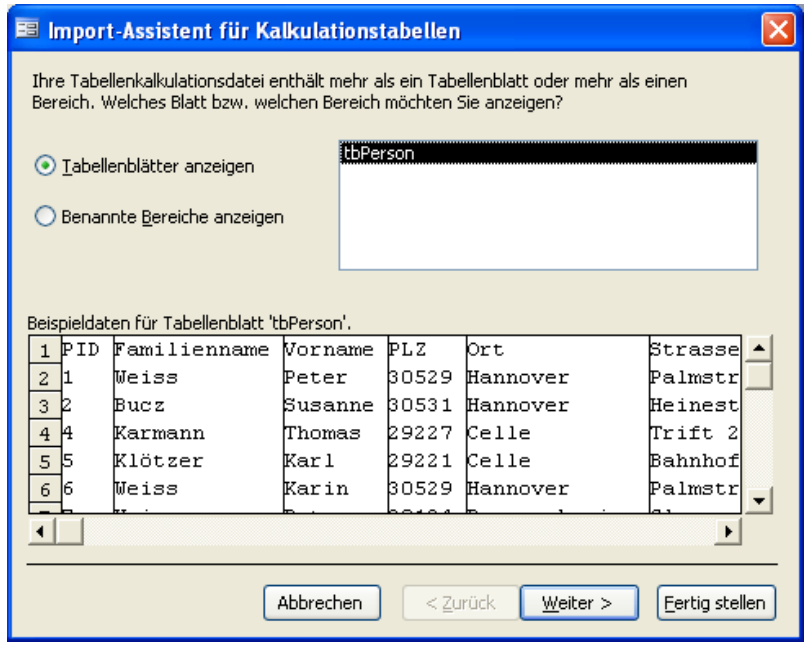


Abbildung 1.32
Auswahl der Excel-Dateien für den Import

5. Wählen Sie die zu importierende Datei (beim ersten Aufbau der Kursdatenbank ist das *Person*) und bestätigen Sie Ihre Auswahl mit der Schaltfläche IMPORTIEREN. Sie gelangen in den sogenannten Import-Assistenten.
6. Bestätigen Sie die Option TABELLENBLÄTTER ANZEIGEN (siehe Abbildung A.33). Alternativ könnten Sie hier weitere Tabellenblätter oder benannte Bereiche aus Excel als Datenquelle wählen. Damit können gezielt bestimmte Daten in MS Access überführt werden.

Abbildung A.33
Import-Assistent Schritt 1


Import-Assistent für Kalkulationstabellen
Ihre Tabellenkalkulationsdatei enthält mehr als ein Tabellenblatt oder mehr als einen Bereich. Welches Blatt bzw. welchen Bereich möchten Sie anzeigen?
Tabellenblätter anzeigen
Benannte Bereiche anzeigen
tbPerson
Beispieldaten für Tabellenblatt 'tbPerson'.
1 PID Familienname Vorname PLZ Ort Strasse
2 1 Weiss Peter 30529 Hannover Palmstr
3 2 Bucz Susanne 30531 Hannover Heinest
4 4 Karmann Thomas 29227 Celle Trift 2
5 5 Klötzer Karl 29221 Celle Bahnhof
6 6 Weiss Karin 30529 Hannover Palmstr
Abbrechen
< Zurück
Weiter >
Fertig stellen

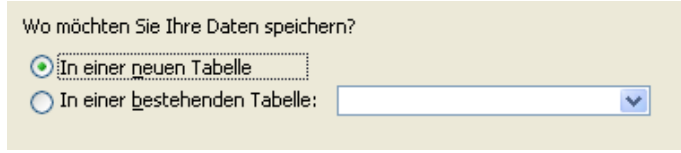
7. Bestätigen Sie im nächsten Schritt, dass die erste Zeile die Spaltenüberschriften bestimmen soll (siehe Abbildung A.34). Damit wird die erste Zeile der Excel-Datei genutzt, um die Namen der Daten in den darauffolgenden Zeilen für die neue MS Access-Tabelle festzulegen. Die erste Zeile bestimmt die Tabellenstruktur, die weiteren Zeilen liefern die eigentlichen Daten. Damit ersparen Sie sich zunächst viel „Tipparbeit“.

Abbildung A.34
Die erste Excel-Zeile definiert die Attributnamen.

Microsoft Access kann Ihre Spaltenüberschriften als Feldnamen verwenden. Enthält die erste angegebene Zeile Spaltenüberschriften?
Erste Zeile enthält Spaltenüberschriften

8. Behalten Sie im nächsten Schritt die Option In einer neuen Tabelle bei (siehe Abbildung A.35). Damit wird das SQL für die Erstellung einer neuen Tabelle generiert und diese anschließend mit Daten gefüllt. Der Import in eine bestehende Tabelle kann in vielen Fällen an veränderten Strukturen, doppelten Schlüsseln und anderen Problemen scheitern. Daher sollte später bei Reparaturen und weiteren Importen stets geprüft werden, ob die Strukturen wirklich identisch sind.

Abbildung A.35
Daten sollten in eine neue Tabelle importiert werden.



9. Im nächsten Schritt hat MS Access die Daten ab Zeile 2 analysiert und daraus Folgerungen zum Datentyp gezogen (siehe Abbildung 1.36). Typische Werte sind „Double“ für Zahlen, „Text“ für alphanumerische Felder und „Datum/Uhrzeit“, wenn Access charakteristische Formate für eine Datums- und/oder Uhrzeitangabe erkannt hat. Sie können durch Markieren der Spalten die erkannten Werte prüfen. Sie können auch eine Spalte nach der Markierung vom Import ausschließen und so eine Auswahl der gewünschten Felder treffen. Das macht immer dann Sinn, wenn Sie es mit denormalisierten komplexen Dateien zu tun haben, die auf verschiedene Tabellen zu verteilen sind, siehe hierzu die Kapitel 7 und 8. Übernehmen Sie die Vorgaben von MS Access.

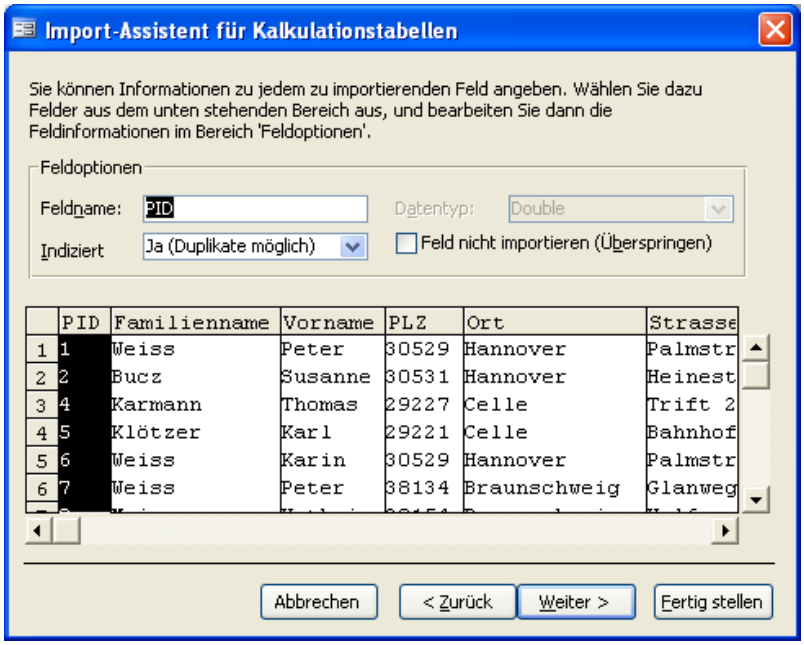
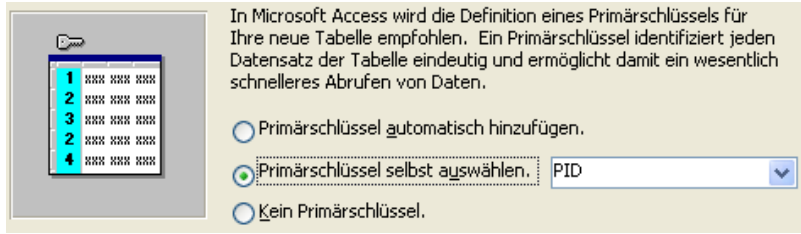

Import-Assistent für Kalkulationstabellen
Sie können Informationen zu jedem zu importierenden Feld angeben. Wählen Sie dazu Felder aus dem unten stehenden Bereich aus, und bearbeiten Sie dann die Feldinformationen im Bereich 'Feldoptionen'.
Feldoptionen
Feldname: PID
Datentyp: Double
Indiziert Ja (Duplikate möglich)
Feld nicht importieren (Überspringen)
PID Familienname Vorname PLZ Ort Strasse
1 1 Weiss Peter 30529 Hannover Palmstr
2 2 Bucz Susanne 30531 Hannover Heinest
3 4 Karmann Thomas 29227 Celle Trift 2
4 5 Klötzer Karl 29221 Celle Bahnhof
5 6 Weiss Karin 30529 Hannover Palmstr
6 7 Weiss Peter 38134 Braunschweig Glanweg
Abbrechen
< Zurück
Weiter >
Fertig stellen

Abbildung 1.36
Automatische Festlegung des Datentyps und der Indizierung

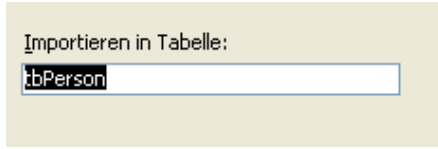
10. Im nächsten Schritt ist der Primärschlüssel für die neue Tabelle zu bestimmen. MS Access schlägt als Standard immer ein eigenes zusätzliches Attribut vor, das erzeugt werden soll und in dem eine fortlaufende Nummerierung eingeführt werden soll. Sie sollten bei allen Tabellen des Beispiels „Kurse“ immer die erste Spalte als selbst ausgewählten Primärschlüssel einstellen, da alle Dateien entsprechend vorbereitet sind (siehe Abbildung A.37).

Abbildung A.37
Auswahl eines Primärschlüssels


In Microsoft Access wird die Definition eines Primärschlüssels für Ihre neue Tabelle empfohlen. Ein Primärschlüssel identifiziert jeden Datensatz der Tabelle eindeutig und ermöglicht damit ein wesentlich schnelleres Abrufen von Daten.
Primärschlüssel automatisch hinzufügen.
Primärschlüssel selbst auswählen.
PID
Kein Primärschlüssel.

11. Schließlich wird im letzten Schritt die Tabelle benannt. Hier können Sie jeweils den vorgegebenen Dateinamen aus Excel verwenden (siehe Abbildung A.38). Natürlich können Sie auch eigene Tabellennamen vergeben. In MS Access hat sich eingebürgert, dass man Präfixe verwendet. Im Datenbankfenster haben Sie bereits gesehen, dass es neben Tabellen Abfragen, Formulare, Berichte, Seiten, Makros, Module und Gruppen gibt. Bei komplexen Anwendungen kann es hier schnell zu Verwechslungen kommen. Da Tabellen die einzigen originären Datenbankobjekte darstellen, die die eigentlichen Daten enthalten, nehmen sie eine Sonderrolle ein. In MS Access geht das in der Fülle der Objekte oftmals leider etwas verloren.

Abbildung A.38
Benennung der Tabelle und Fertigstellung



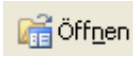
Sie erhalten jetzt eine Bestätigungsmeldung von MS Access. Danach sehen Sie im Datenbankfenster, dass jetzt als erste Tabelle die Tabelle *tbPerson* eingetragen worden ist.

Import der Tabellen der Kursdatenbank

Sie haben also als Erstes die Tabelle *tbPerson* aus einer Excel-Datei erzeugt. Mit demselben Verfahren können Sie jetzt die weiteren Tabellen importieren. Importieren Sie für die Kursdatenbank die folgenden fünf Tabellen:

- Person
- Kursbesuche
- Dozent
- Kursthema
- Kurs

Die weiteren Dateien sind Testdateien für spezielle Beispiele, die Sie auch bei Bedarf später noch ergänzen können.



Sie sollten Ihren Import noch kurz prüfen, bevor Sie die weiteren Schritte durchführen. Markieren Sie nacheinander jeweils eine Tabelle und öffnen Sie sie im Datenbankfenster mit Doppelklick oder über die Schaltfläche ÖFFNEN.

Prüfen Sie den Inhalt. Sie sollten in `tbDozent` 5 Datensätze, in `tbKurs` 7 Datensätze, in `tbKursbesuche` 18 Datensätze, in `tbKursthema` 11 Datensätze

und in `tbPerson` 20 Datensätze finden. Wenn Sie den Inhalt genau überprüfen wollen, sollten Sie Ihre Datenbankinhalte mit den im Anhang abgedruckten Tabellen vergleichen.

A.5 openBase

A.5.1 SQL-Anweisungen eingeben

openBase bietet als zweite Datenbank eine grafische Oberfläche mit einer ähnlichen Struktur wie MS Access. Im linken Bereich des Datenbankfensters befinden sich die Datenbankobjekte. Wieder beginnt die Aufzählung mit den Tabellen, die das Kernstück der Datenbank bilden und der Speicherung der Daten dienen. Dann folgen die Abfragen. Wird die Schaltfläche ABFRAGEN gewählt, werden im rechten Teil des Datenbankfensters die möglichen Aktionen, wie in Abbildung 1.39 wiedergegeben, dargestellt.

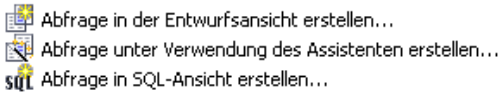

Abfrage in der Entwurfsansicht erstellen...
Abfrage unter Verwendung des Assistenten erstellen...
Abfrage in SQL-Ansicht erstellen...

Abbildung 1.39
Mögliche Aktionen für Abfragen

Alle drei Möglichkeiten führen im Ergebnis zu einer SQL-Anweisung, nur die Wege sind unterschiedlich.

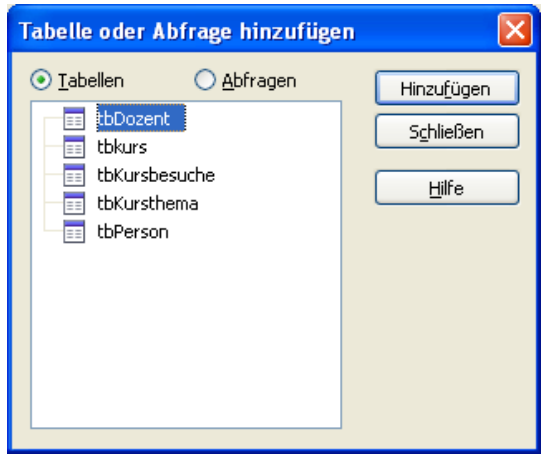
- Die Entwurfsansicht ist eine grafische Oberfläche, in der die Abfrage zusammengestellt und dann daraus die SQL-Anweisung generiert wird.
- Mit den Assistenten wird noch ein Abfragedialog vor die Entwurfsansicht und die SQL-Generierung geschaltet. Somit gibt es hier drei Stufen: Dialog, Entwurf und schließlich SQL.
- Mit der SQL-Ansicht schließlich kann eine SQL-Anweisung direkt eingegeben werden.

Für unsere Zwecke sind die Assistenten wenig hilfreich. Die Entwurfsansicht kann in einigen Fällen dazu verwendet werden, um das Ergebnis als SQL-Anweisung zu analysieren. Sie soll daher hier einmal beispielhaft aufgerufen werden. openBase fragt dann nach den zu verwendenden Datenquellen, wie in Abbildung A.40 zu sehen.

Wählen Sie beispielsweise *tbPerson* aus. Sie gelangen dann in das Übersichtsfenster für die Erstellung einer Abfrage (siehe Abbildung A.41). Im oberen Teil des Fensters sind die ausgewählten Datenquellen dargestellt. Darunter ist eine tabellenartige Struktur, die der Konstruktion der Abfrage und schließlich der SQL-Generierung dient.

Wie in MS Access können Sie die gewünschten Datenfelder entweder per Doppelklick auf die Datenfelder im oberen Tabellenbereich oder mit Ziehen mit gedrückter linker Maustaste aus dem oberen in die Tabelle im unteren Bereich in die Abfrage einfügen.

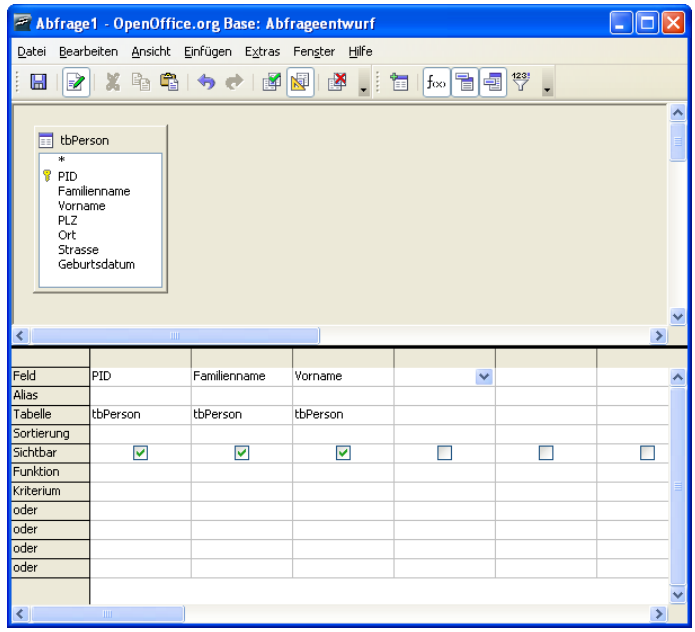
Abbildung A.40
Datenquellen für die Abfrage



Eine Besonderheit von openBase ist dabei, dass nicht nur die Namen der Felder aus der Datenbank direkt, sondern auch sogenannte Alias verwendet werden dürfen. Alias werden von allen größeren Datenbanksystemen unterstützt. openBase hat hier eine elegante Möglichkeit gefunden, sie auch unmittelbar in die grafische Oberfläche zu integrieren.

Außerdem lassen sich die ausgewählten Felder zur Sortierung der Ergebnisdatensätze, zur Auswahl der relevanten Datensätze (Kriterien) sowie zur Gruppierung mit Funktionen heranziehen. Auf die Bedeutung dieser Optionen und ihre Auswirkung auf die generierte SQL-Anweisung wird im Buchtext eingegangen.

Abbildung A.41
Übersichtsfenster für die Erstellung einer Abfrage in openBase



Haben Sie die Abfrage in der Entwurfsansicht erstellt, können Sie sie unmittelbar speichern. Dies sollten Sie in regelmäßigen Abständen tun. Wenn Sie, wie auch in Abbildung A.44 dargestellt, nach einem Namen gefragt werden, handelt es sich um den Namen, unter dem die Abfrage in der Übersicht des Datenbankfensters dargestellt werden soll. Gespeichert wird immer die Abfrage, nicht die mit der Abfrage ermittelten Daten.

Sie können aus der Entwurfsansicht jederzeit in die SQL-Ansicht wechseln. Dazu müssen Sie nur das in der Symbolleiste aufgeführte und hier angegebene Symbol wählen und ausschalten. Nach dem Ausschalten der Entwurfsansicht arbeitet openBase grundsätzlich in der SQL-Ansicht. Sie können die generierte SQL-Anweisung sehen, wie in Abbildung 1.42 dargestellt. Das Umschalten zwischen Entwurfsansicht und SQL-Ansicht funktioniert in beiden Richtungen, soweit es sich um eine gültige und darstellbare SQL-Anweisung handelt. Also auch, wenn Sie zu Beginn die SQL-Ansicht gewählt haben, können Sie wieder in die Entwurfsansicht zurückwechseln.

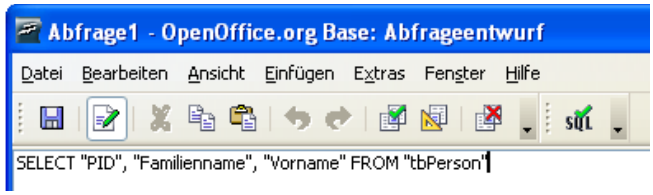

Abfrage1 - OpenOffice.org Base: Abfrageentwurf
Datei Bearbeiten Ansicht Einfügen Extras Fenster Hilfe
SELECT "PID", "Familienname", "Vorname" FROM "tbPerson"

Abbildung 1.42
SQL-Anweisung in der SQL-Ansicht

Bisher haben wir nur auf verschiedene Arten die SQL-Anweisung formuliert. Sie wollen sie natürlich auch ausführen, um die Ergebnisdaten zu erhalten. Dafür können Sie das am Seitenrand angegebene Symbol verwenden. Das Ergebnis ist in Abbildung 1.43 zu sehen.

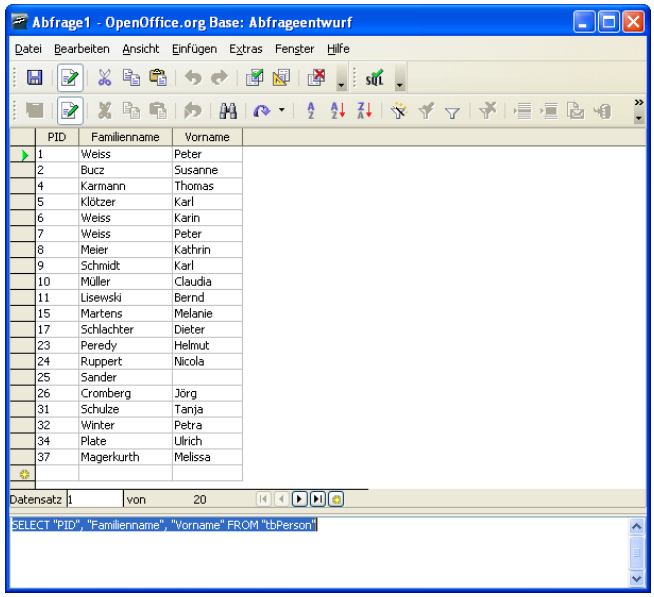
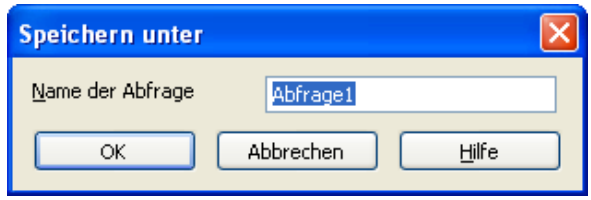


Abbildung 1.43
Ergebnisdatensätze mit SQL-Anweisung (markiert)

Sie sehen, dass openBase neben den Ergebnissen auch die SQL-Anweisung – beziehungsweise bei eingeschalteter Entwurfsansicht auch diese – dargestellt.

Abbildung A.44
Benennung der Abfrage



Sie können die erstellte Abfrage jetzt beliebig ändern und ergänzen. Sie lässt sich jeweils wieder erneut ausführen und somit auf die gewünschten Ergebnisse testen. Speichern Sie die Abfragen dieses Buches unter den Namen, die Ihnen sinnvoll erscheinen, wobei Sie auch die Seitennummern mit einbeziehen können.

Haben Sie eine Abfrage einmal gespeichert, wird sie in der Übersichtsliste des Datenbankfensters dargestellt. Benennen Sie sie beispielsweise mit „Testabfrage“, erscheint dieser Name im Fenster, wie in Abbildung A.45 dargestellt.

Abbildung A.45
Datenbankfenster mit Abfrage

Eine Besonderheit von openBase ist, dass rechts neben dieser Liste das Ergebnis der Abfrage dynamisch eingeblendet werden kann. Wechseln Sie dazu gegebenenfalls von der Ansicht KEINE in die Ansicht DOKUMENT,

Das Datenbanksystem openBase bietet im Rahmen seiner grafischen Benutzeroberfläche die Möglichkeit, Alias für die Felder einer Abfrage festzulegen. Dazu müssen Sie das nebenstehende Symbol aktivieren. Sie erhalten dann eine weitere Zeile, in der Sie unmittelbar ein Alias für ein Feld angeben können.

Darüber hinaus können Sie auch über das nebenstehende Symbol bestimmen, dass nur unterschiedliche Datensätze in der Ergebnismenge enthalten sein sollen. Es wird dann ein `DISTINCT` in die entsprechende SQL-Anweisung generiert.

A.5.2 Andere SQL-Anweisungen eingeben

Die Abfragen erlauben in openBase letztlich nur die Eingabe von SELECT-Anweisungen, was systematisch auch richtig ist. Andere SQL-Anweisungen, also INSERT, UPDATE und DELETE sowie alle SQL-DDL- und SQL-DCL-Anweisungen, lassen sich direkt als SQL-Anweisungen eingeben.

Wählen Sie im Menü EXTRAS/SQL, um in ein Eingabefenster für SQL-Anweisungen zu gelangen (siehe Abbildung 1.46).

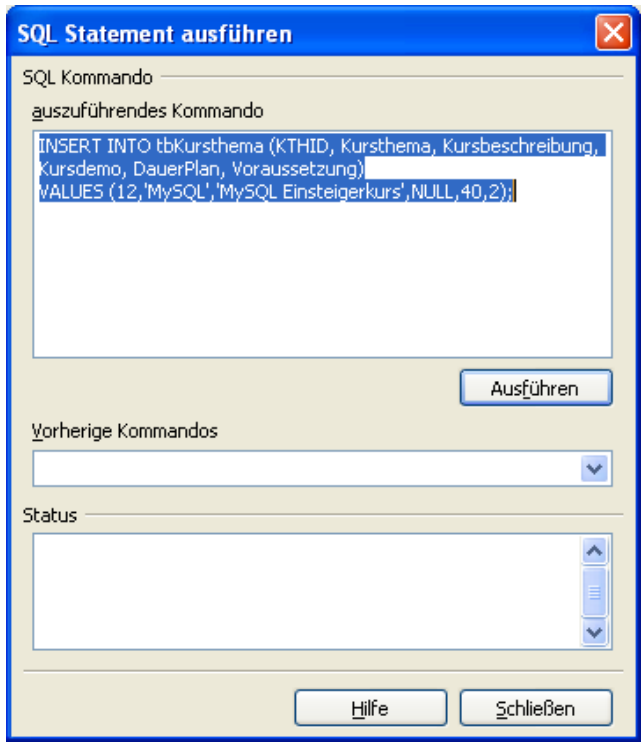


Abbildung 1.46
Direkte SQL-Eingabe in openOffice.base

Hier können Sie alle Anweisungen außer SELECT-Anweisungen eingeben. Aber Vorsicht, openBase ist hier einmal mehr recht sensibel. Tabellennamen und Feldnamen ohne Anführungsstriche werden in Großschreibung umgesetzt, wie es dem SQL-Standard entspricht. Wenn Sie in Ihrer Datenbank aber die Tabellen oder Felder in gemischter Schreibweise benannt haben, erkennt openBase diese nicht mehr. Sie müssen also Anführungsstriche verwenden, wie im Beispiel in Listing 1.1 zu sehen. Wenn Sie ganz sichergehen wollen, setzen Sie alles in Anführungsstriche und achten auf eine korrekte Schreibweise auch hinsichtlich der Groß- und Kleinschreibung.

```
INSERT INTO "tbKursthema" (KTHID, "Kursthema", "Kursbeschreibung",
"Kursdemo", "DauerPlan", "Voraussetzung")
VALUES (12,'MySQL','MySQL Einsteigerkurs',NULL,40,2);
```

Listing 1.1
Beispiel eines SQL-Befehls in openBase

Zum Abschluss noch ein Hinweis: openBase verwendet normalerweise die Datenbankmaschine HSQL zur Speicherung der Daten und Ausführung der SQL-Anweisungen. openBase verfolgt aber einen breiteren Ansatz als andere Systeme und stellt seine Oberfläche auch anderen Datenbanksystemen wie MySQL „zur Verfügung“, oder anders ausgedrückt, openBase kann auf Daten in diesen Systemen zugreifen und SQL-Befehle ausführen.

Bereits beim Start der Oberfläche fragt Sie der Assistent, ob Sie eine neue Datenbank erstellen, eine bestehende Datenbankdatei öffnen oder die Verbindung zu einer bestehenden Datenbank herstellen wollen.

Im ersten Fall wird eine neue ODS-Datei angelegt, die tatsächlich auch die Daten Ihrer Datenbank speichert und mit der HSQL-Datenbankmaschine genutzt werden kann.

Im zweiten Fall wird eine bestehende ODS-Datei geöffnet.

Im dritten Fall können Sie eine Verbindung zu einer anderen bestehenden Datenbank herstellen. Hier stehen beispielsweise auch MS Access, MySQL und ein Oracle-Zugriff über JDBC oder ODBC zur Verfügung, sodass diese Systeme genutzt werden können.

Weitere Informationen über die javabasierte HSQL-Engine finden Sie im Internet unter diesem Stichwort, insbesondere auf *www.hsql.org*.

B

Anhang: Boolesche Algebra

SQL verwendet an verschiedenen Stellen Ausdrücke, die einen sogenannten Wahrheitswert liefern, also wahr oder falsch sind. Dies gilt beispielsweise für die `WHERE`-Klausel der `SELECT`-Anweisung. Diese Ausdrücke können mit den typischen Operatoren UND und ODER kombiniert werden, in einigen Datenbanksystemen auch mit weiteren Operatoren.

Daher sollen hier die wesentlichen Grundlagen der Logik erläutert werden.

Grundlegend ist zunächst der Begriff der Aussage. Eine Aussage ist ein Ausdruck, für den eindeutig entscheidbar ist, ob er wahr oder falsch ist. Beispiele sind

A: Die Zahl 6 ist eine gerade Zahl. (wahr)

B: Berlin liegt in Frankreich. (falsch)

Eine Aussage wird im Folgenden mit Buchstaben wie A oder B bezeichnet. Es existieren Operatoren, mit denen Aussagen verknüpft werden können. Die bekanntesten Operatoren sind UND (Konjunktion) und ODER (Disjunktion).

UND

So gilt für zwei Aussagen A und B, dass die Gesamtaussage (A UND B) nur wahr ist, wenn beide Aussagen wahr sind.

A: Die Zahl X ist durch 2 teilbar.

B: Die Zahl X ist durch 5 teilbar.

C: Die Zahl X ist durch 10 teilbar (A UND B).

Wenn A UND B für eine Zahl X gilt, dann gilt auch, dass X durch 10 teilbar ist.

Beispiele:

X = 30: A ist wahr, B ist wahr, also ist A UND B wahr (30 ist durch 10 teilbar).

X = 15: A ist falsch, B ist wahr, also ist A UND B falsch (15 ist nicht durch 10 teilbar).

X = 12: A ist wahr, B ist falsch, also ist A UND B falsch (12 ist nicht durch 10 teilbar).

X = 11: A ist falsch, B ist falsch, also ist A UND B falsch (11 ist nicht durch 10 teilbar).

Inklusive-ODER

Entsprechend gilt für zwei Aussagen A und B, dass die Gesamtaussage (A ODER B) wahr ist, wenn eine (oder beide) Aussagen wahr sind. Da auch beide Aussagen wahr sein können, spricht man von einem Inklusive-ODER (einschließlich zweier richtiger Aussagen).

A: Die Sonne scheint.

B: Das Licht ist an.

C: Es ist hell (A ODER B).

Wenn A ODER B wahr ist, dann gilt auch C, dass es hell ist.

Sonne scheint, Licht an: A ist wahr, B ist wahr, also ist A ODER B wahr (hell).

Sonne scheint nicht, Licht an: A ist falsch, B ist wahr, also ist A ODER B wahr (hell).

Sonne scheint, Licht aus: A ist wahr, B ist falsch, also ist A ODER B wahr (hell).

Sonne scheint nicht, Licht aus: A ist falsch, B ist falsch, also ist A ODER B falsch (nicht hell).

Exklusive-ODER

Die dritte Kombination für zwei Aussagen A und B ist, dass die Gesamtaussage (A ODER B) wahr ist, wenn genau eine (nicht beide) Aussage wahr ist. Da nur eine der beiden Aussagen wahr sein darf, spricht man von einem Exklusive-ODER.

Beispiel

A: Die Zahl X ist negativ.

B: Die Zahl Y ist negativ.

C: Das Produkt X*Y ist negativ. (A XOR B).

Wenn A XOR B wahr ist, dann gilt auch C, dass das Produkt negativ ist.

X=-2, Y=-3: A ist wahr, B ist wahr, A XOR B ist falsch (das Produkt ist positiv).

X=+2, Y=-3: A ist falsch, B ist wahr, A XOR B ist wahr (das Produkt ist negativ).

X=-2, Y=+3: A ist wahr, B ist falsch, A XOR B ist wahr (das Produkt ist negativ).

X=+2, Y=+3: A ist falsch, B ist falsch, A XOR B ist falsch (das Produkt ist positiv).

Die Tabelle B.1 zeigt die Zusammenhänge.

A	B	A UND B	A ODER B	A XOR B
w	w	w	w	f
w	f	f	w	w
f	w	f	w	w
f	f	f	f	f

Tabelle B.1
Wahrheitstafel

Neben diesen Zusammenhängen gibt es noch die Negation NICHT. NICHT verändert ein „wahr“ in ein „falsch“ und umgekehrt.

C

Anhang: Daten

C.1 Datenbank Kurse

PID	Familienname	Vorname	PLZ	Ort	Strasse	Geburtsdatum
1	Weiss	Peter	30529	Hannover	Palmstraße 6	07.11.1963
2	Bucz	Susanne	30531	Hannover	Heinestraße 23	06.04.1976
4	Karmann	Thomas	29227	Celle	Trift 28	04.08.1954
5	Klötzer	Karl	29221	Celle	Bahnhofstraße 2	13.03.1971
6	Weiss	Karin	30529	Hannover	Palmstraße 6	05.10.1962
7	Weiss	Peter	38134	Braunschweig	Glanweg 4	02.03.1974
8	Meier	Kathrin	38154	Braunschweig	Welfenallee 23	03.05.1981
9	Schmidt	Karl	30529	Hannover	Lavesallee 32	25.06.1949
10	Müller	Claudia	29596	Breitenhees	In den Fuhren 12	
11	Lisewski	Bernd	30890	Barsinghausen	Roggenkamp 10	06.06.1960
15	Martens	Melanie	29221	Celle	Horstweg 258	17.02.1961
17	Schlachter	Dieter	29227	Celle	Mondhagen 43	02.02.1961
23	Peredy	Helmut	29221	Celle	Mauernstraße 2	23.02.1956
24	Ruppert	Nicola	29301	Bergen	Welfenallee 23	25.02.1962
25	Sander		29223	Celle	Marxallee 12	05.02.1953
26	Cromberg	Jörg	38152	Braunschweig	Nordring 13	07.06.1991
31	Schulze	Tanja	29308	Winsen	Berliner Ring 23	09.11.1992
32	Winter	Petra	29320	Hermannsburg	Immenhoop 51	30.12.1989
34	Plate	Ulrich	30529	Hannover	Gutenberggasse 5	02.12.1986
37	Magerkurth	Melissa	29336	Nienhagen	Am Tümpel 3	04.09.1951

Abbildung C.1
Tabelle tbPerson

DID	PID	Beschaeftigungsbeginn	Stundensatz	Firma	Titel	Qualifikationen
812	1	01.07.2003	17,00 €	selbstständig	Informatiker	Word, Windows, Datenbanken
815	17	01.01.2002	15,00 €	selbstständig	IT-Kaufmann	Office, Datenbanken, Netzwerke
821	6	15.09.2005	13,00 €	Dreher KG	Meister	CNC-Steuerungen
833	26	01.08.2003	14,00 €	Pensionär	Mathematiker	Windows, Word
834	2	01.01.2001	15,00 €	Paulsen GmbH	Industriekaufmann	Windows, Rechnungswesen

Abbildung C.2
Tabelle tbDozent

Abbildung C.3
Tabelle tbKursthema

KTHID	Kursthema	Kursbeschreibung	Kursdemo	DauerPlan	Voraussetzung
1	Einstieg mit Windows	Windows bietet die Grundlage für alle modernen und		20	
2	Windows für Fortgeschrittene	Der Kurs wendet sich an PC-Benutzer, die schon ewe		20	1
3	Ordnung im Dateisystem	Wie schaffe ich Ordnung auf dem PC? Wie können F		8	2
4	Word I	Einstieg in die Textverarbeitung mit Word. Grundlegı		40	2
5	Access I	Einstieg in das Datenbankmanagementsystem Acc		40	2
6	Excel I	Einstieg in die Tabellenkalkulation mit Excel. Grundf		40	2
7	Access II	Vertiefung der Arbeit mit MS Access. Komplexe Abfr		40	5
8	Datenbankentwurf	Entwurf relationaler Datenbanken und Umsetzung in		80	7
9	VBA-Datenbankprogrammierung	Programmierung von Anwendungen auf der Basis v		80	7
10	Word II	Vertiefung des Umgang mit Word. Mehrspaltige Forı		40	4
11	Excel II	Vertiefung des Umgangs mit Excel. Funktionen einsc		40	6

Abbildung C.4
Tabelle tbKurs

KID	Kurskennung	KTHID	KursdauerStunden	Kursbeginn	Kursende	Zertifikat	Gebuehr	Ustpflichtig	DID
CE17	Celle17-Word	4	40	23.04.2007	27.04.2007	eigen	280,00 €	WAHR	833
CE20	Celle20-Word	4	40	09.07.2007	13.07.2007	eigen	280,00 €	WAHR	812
CE23	Celle23-Access	5	36	06.08.2007	10.08.2007	eigen	350,00 €	FALSCH	812
CE24	Celle24-Access	7	40	13.08.2007	17.08.2007	eigen	400,00 €	FALSCH	812
H89	Hannover89-Access	5	32	09.07.2007	13.07.2007	eigen	350,00 €	FALSCH	812
H90	Hannover90-Windows	1	18	16.07.2007	18.07.2007	kein	120,00 €	WAHR	834
H91	Hannover91-Windows	2	18	23.07.2007	25.07.2007	kein	120,00 €	WAHR	834

Abbildung C.5
Tabelle tbKursbesuche

KBID	KTID	KID	Fehltage	Zeugnis	Selbstzahler	Zahlweise	Rabatt	GezahlterBetrag
310	25	CE17	0	WAHR	WAHR	Überweisung	20,00 €	260,00 €
312	4	CE24	0	FALSCH	FALSCH	Gutschein	100,00 €	250,00 €
314	4	CE17	0	WAHR	FALSCH	Gutschein	0,00 €	280,00 €
315	5	CE17	0	WAHR	FALSCH	Gutschein	0,00 €	280,00 €
354	34	H90	1	FALSCH	WAHR	Überweisung	20,00 €	80,00 €
361	9	H90	0	FALSCH	FALSCH	Gutschein	0,00 €	120,00 €
412	4	CE23	2	FALSCH	FALSCH	Gutschein	0,00 €	350,00 €
413	5	CE23	0	FALSCH	WAHR	Überweisung	100,00 €	50,00 €
415	15	CE23	5	FALSCH	WAHR	Bar	100,00 €	250,00 €
416	10	CE23	0	FALSCH	FALSCH	Gutschein	0,00 €	350,00 €
418	31	CE23	1	FALSCH	FALSCH	Gutschein	0,00 €	350,00 €
419	32	CE23	0	FALSCH	WAHR	Überweisung	50,00 €	300,00 €
420	24	CE23	2	FALSCH	WAHR	Bar	0,00 €	350,00 €
421	37	CE23	8	FALSCH	WAHR	Überweisung	150,00 €	50,00 €
423	10	CE24	0	FALSCH	WAHR	Bar	100,00 €	0,00 €
432	15	CE24	0	FALSCH	WAHR	Bar	100,00 €	0,00 €
438	11	CE24	0	FALSCH	FALSCH	Gutschein	0,00 €	350,00 €
442	23	CE24	0	FALSCH	WAHR	Überweisung	50,00 €	300,00 €

C.2 Datenbank Artikel

MNr	KdNr	Firma	Anrede	Name	Vorname	Strasse	PLZ	Ort	Geburtsdatum
1	1		Frau	Sibum	Jutta	Hannoversche Straße 6	29223	Celle	02.04.65
1	2	Ewald Kuhn KG	Herr	Schöller	Jürgen	Windmühlenstraße 12	29233	Celle	04.07.43
1	3			Bit	Hans-Joachim	Lange Straße 15	29320	Hermannsbur	30.06.68
2	4	Franz Koch	Frau	Lange	Rosemarie	Friedrichstraße 28	26203	Wardenburg	31.10.78
1	5		Herr	Thole	Eckhard	Breslauer Straße 15	26203	Wardenburg	23.11.69
1	6	Hobelix OHG	Herr	Falk	Hermann	Lange Laube 2	30230	Hannover	05.02.52
2	7		Frau	Kurz	Claudia	Tulpenstraße 25	26169	Friesoythe	27.04.56
3	8		Herr	Karl	Oliver	Lange Straße 15	29331	Lachendorf	18.07.64
1	9	Schütte & Karlsen	Frau	Karlsen	Manuela	Hauptstraße 64	26132	Oldenburg	23.07.65
1	10	Wilhelm Kuhl KG	Herr	Kuhl	Rolf	Am Wasser 29	26683	Strücklingen	09.02.56
2	11		Herr	Hoppe	Heinz	Am Lohbusch 2	29221	Celle	28.03.57
1	12		Frau	Kohlmeier	Gerda	Goetheweg 5	29331	Lachendorf	22.03.39
2	13			Körber	Margarete	Am Rübenberg 19	29331	Lachendorf	02.08.52
2	14		Herr	Maischeck	Gunter	Marktstraße 3	29331	Lachendorf	31.10.73
1	15		Frau	Mathes	Gundula	Schillergasse 8	29331	Lachendorf	01.05.34
1	16	Kuhlmann & Sohn	Herr	Meyer	Helmut	Sonnenweg 13	29223	Celle	11.09.56
1	17		Frau	Michel	Annette	Apfelweg 8	29225	Celle	20.04.46
1	18		Herr	Timme	Ulf	Geschwister-Scholl-Straße 117	29223	Celle	23.12.71
1	19		Herr	Thiermann	Peter	Trinkhorstweg 3	29223	Celle	27.12.81
2	20		Herr	Tiegs	Andreas	Oesterleystraße 1	29225	Celle	03.02.54
1	21	Buchholz	Herr	Tisson	Thomas	Neustadt 17	29221	Celle	29.04.76
1	22		Herr	Kohlbausch	Hans	Prinzengarten 5	29223	Celle	02.06.56
1	23		Herr	Überholz	Matthias	Lindenstraße 4	29227	Celle	05.08.85
2	24		Frau	Petersen	Annemarie	Neustadt 3	29221	Celle	05.08.34
1	25	Clean GmbH	Frau	Havenstein	Katrin	Lauensteinplatz 8	29225	Celle	03.03.86
2	26		Herr	Sülzer	Bernd	Nordweg 27	29229	Celle	15.04.83
3	27		Frau	Jakobi	Irmgard	Vogelbeerweg 3	29223	Celle	03.04.62
1	28		Frau	Hoopmann	Kirsten	Akazienweg 45	29221	Celle	27.02.87
2	29		Herr	Roberts	Oliver	Jägerheide 3	29227	Celle	23.03.39
1	30		Herr	Meyer	Ewald	Am Weinberg 15	20221	Celle	21.11.58
1	31		Herr	Meyer	Helmut	Sonnenweg 13	29223	Celle	11.09.56
1	32		Herr	Maier	Siegfried	Kampweg 3	29303	Bergen	01.07.60
2	33		Frau	Rabiga	Annelore	Dorfstraße 17	29303	Bergen	23.05.51
1	34	Bohlen & Co.	Frau	Pfaff	Else	Örtzestraße 5	29303	Bergen	19.12.37
2	35		Frau	Pfaff	Hertha	Kreuzweg 8	29303	Bergen	12.01.41
3	36		Frau	Mengel	Margrit	Rosenstraße 8	29303	Bergen	30.09.49
2	37		Herr	Kramm	Heiko	Luxhoop 5	29303	Bergen	07.07.88
1	38		Herr	Förster	Ernst-Jürgen	Achtergarten 1	29348	Eschede	13.10.46
1	39		Herr	Dreyer	Manfred	Thieleweg 3	29348	Eschede	23.03.62
2	40		Frau	Förster	Iris	Salzmoor 11	29328	Faßberg	17.06.69
1	41		Herr	Wieckenberg	Peter	Wieseweg 21	29328	Faßberg	06.04.56
3	42		Herr	Stechert	Otto	Am Gehäge 2	29328	Faßberg	26.04.31
1	43		Frau	Krenz	Ingrid	Ericaweg 17	29313	Hambühren	09.09.51
2	44		Frau	Licht	Marion	Beethovenstraße 10	29313	Hambühren	03.04.61
1	45		Herr	Leu	Walter	Birkenweg 9	29320	Hermannsbur	01.08.62
1	46		Herr	Guhr	Bernd	Heidberg 3	29320	Hermannsbur	23.05.48
1	47		Herr	Gertendorf	Rainer	Lotharstraße 49	29320	Hermannsbur	03.01.89
1	48		Herr	Böger	Werner	Willighausen 34	29320	Hermannsbur	22.02.61
1	49	Fabian Bullerich	Herr	Behn	Heinrich	Sägenförth 13	29320	Hermannsbur	23.02.68

Abbildung C.6
Tabelle Kunde Teil 1
(Ausschnitt)

1	50		Frau	Bauer	Tatjana	Dicksberg 1	29320	Hermannsbur	24.09.87
1	51		Frau	Suchy	Jessica	Unter den Eichen 7	29320	Hermannsbur	12.12.51
1	52		Herr	Siegmann	Rudolf	Ericaweg 17	29320	Hermannsbur	10.05.79
1	53	Rein & Sohn	Herr	Schitzke	Michael	Alte Gasse 15	29320	Hermannsbur	31.12.83
2	54		Herr	Skraole	Dietmar	Georg-Haccius-Straße 12	29320	Hermannsbur	03.07.61
1	55		Herr	Tasasche	Gunter	Olendorp 6	29320	Hermannsbur	08.02.41
1	56		Herr	Grambow	Siegfried	Lange Str. 15	29331	Lachendorf	06.08.42
1	57		Herr	Mitzke	Anton	Kurze Str. 12	29331	Lachendorf	23.11.54
1	58		Frau	Dricksen	Anja	Alte Grenze 7	29364	Langlingen	08.05.72
1	59		Frau	Müller	Anna	Alte Grenze 7	29225	Celle	17.03.29
3	60		Frau	Kluge	Jacqueline	Hauptstr. 19	29364	Langlingen	23.08.91
1	61		Herr	Jahnke	Peter	Hauptstraße 22	29364	Langlingen	15.08.65
2	62		Herr	Franck	Bruno	Schwarzer Weg 19	29336	Nienhagen	09.03.46
3	63		Herr	Lahn	Wilhelm	Dorfstraße 9	29336	Nienhagen	09.05.30
1	64		Herr	Reisinger	Thomas	Drosselweg 23	29336	Nienhagen	16.09.81
1	65		Herr	Tölke	Wim	Am Fleitgraben 41	29336	Nienhagen	02.06.38
1	66		Herr	Preißler	Jan-Peter	Dorfstraße 41	29336	Nienhagen	03.06.62
1	67		Herr	Bucco	Arnold	Wiesenstraße 13	29336	Nienhagen	05.03.78
1	68		Frau	Behrens	Cornelia	Maisfeld 32	29336	Nienhagen	06.04.89
1	69		Frau	Bothe	Frieda	Am Bahnhof 3	29339	Wathlingen	04.02.46
1	70		Frau	Bühren	Tamara	Krümmelweg 1	29339	Wathlingen	07.09.87
1	71		Herr	Brase	Jan	Sachsenring 3	29339	Wathlingen	01.02.84
1	72		Herr	Kolditz	Michael	Schaffeld 23	29342	Wienhausen	23.06.78
1	73		Herr	Bühring	Uwe	Schleusenweg 17	29342	Wienhausen	03.07.61
1	74		Herr	Eggers	Harald	Appelweh 4	29342	Wienhausen	08.10.60
1	75		Herr	Büttner	Detlef	Dorfstraße 18	29342	Wienhausen	08.09.58
1	76		Frau	Luft	Carola	Petersweg 20	29323	Wietze	21.11.86
1	77			Lehmann	Dagmar	Friedrichstraße 12	29323	Wietze	20.08.64
1	78		Herr	Müller	Peter	Am Salzberg 9	29323	Wietze	31.12.83
2	79		Herr	Meier	Horst	Buschweg 7	29308	Winsen	06.05.53
2	80		Herr	Meier	Jan-Peter	Bahnhostraße 11	29308	Winsen	02.08.56
1	81		Herr	Langenbahn	Christian	Bremer Weg 17	29308	Winsen	11.07.67
1	82		Herr	Elsner	Rudolf	Hauptstraße 30	29308	Winsen	21.02.61
1	83		Frau	Lüttger	Ingrid	Hirschwiese 19	29308	Winsen	13.07.57
1	84		Frau	Mohrholz	Rosemarie	Poggenteich 31	29223	Celle	05.09.50
1	85		Herr	Meyer	Werner	Heese 10	29348	Eschede	18.04.57
1	86		Herr	Hoffmann	Thomas	Heese 10	29225	Celle	13.03.57
1	87		Herr	Ammermann	Udo	Am Sägewerk 2	29348	Eschede	10.02.61
2	88		Frau	Schröder	Birgit	Harmsstraße 41	29320	Hermannsbur	03.03.67
1	89		Frau	Trester	Erika	Am Hasenberg 2	29320	Hermannsbur	07.08.69
1	90		Herr	Nussbaum	Robert	Ostsiedlung 21	29345	Unterlüß	24.01.48
1	91		Herr	Mohnbaum	Waldemar	Fuchsweg 5	29345	Unterlüß	09.07.67
1	92		Frau	Schilling	Hildegard	Hermannsbuger Straße 3	29345	Unterlüß	03.10.37
1	93		Frau	Schulz	Ute	Südsiedlung 12	29345	Unterlüß	19.08.56
1	94	Schmidt GmbH	Frau	Schmidt	Stephanie	Heidkamp 35	29331	Lachendorf	25.03.61
1	95		Herr	Seidel	Theo	Zugbrückenstraße 13	29221	Celle	06.09.39
1	96		Herr	Ruczynski	Manfred	Neues Land 14	29229	Celle	23.11.61
1	97			Lutz	Thomas	Mauernstraße 32	29221	Celle	02.06.72
1	98		Herr	Scharenberg	Ulrich	Welfenallee 36	29225	Celle	08.12.63
1	99		Herr	Schilling	Lutz	Alte Grenze 37	29221	Celle	03.06.71
2	100		Frau	Nalihn	Bettina	Mühlenstraße 4	29221	Celle	12.02.61

Abbildung C.7
Tabelle Kunde Teil2
(Ausschnitt)

Abbildung C.8
Tabelle mwst

mwst_art	mwst_satz
ermässigt	7
kein	0
voll	19

Abbildung C.9
Tabelle Warengruppe

kennziffer	warengruppe
1	Papier- und Papierprodukte
2	Drucksachen
3	Versandausstattung
4	Schreibmittel
5	Formulare, Blöcke, Klebe- und Haftmittel
6	Ordnungsmittel
7	Büromöbel
8	Präsentationsmittel
9	EDV und sonstige Büroausstattung

Abbildung C.10
Tabelle Artikel

anr	bezeichnung	gebinde	Einheit	kennziffer	einstandspreis	listenpreis	mwst_art	bestand
1002	Universalpapier Standard	500	Blatt	1	2,48	5,99	voll	2000
1123	Kassenrollen	1	Pack	1	0,87	1,39	voll	300
2120	Briefumschläge	1000	Stück	3	8,45	13,79	voll	126
2132	Versandtaschen	250	Stück	3	8,45	13,99	voll	23
2231	Luftpolstertaschen	10	Stück	3	1,02	1,56	voll	2
4323	Druck-Kugelschreiber	1	Stück	4	0,15	0,23	voll	389
4324	Fine-Pen	1	Stück	4	0,18	0,36	voll	231
4468	Druckbleistift	1	Stück	4	0,25	0,39	voll	490
5342	Zettelklotz	250	Blatt	5	1,89	3,59	voll	1230
5346	Notizblöcke	10	Stück	5	3,34	5,44	voll	340
5349	Collegeblock	1	Block	5	0,56	1,59	voll	145
6001	Standardordner	1	Stück	6	1,56	2,49	voll	1430
9056	USB-Stick	2	GB	9	8,89	14,99	voll	12
9058	CD	10	Stück	9	2,11	5,49	voll	342
9801	Kaffee mild	1	kg	9	3,34	9,99	ermässigt	45
9810	Menthol weiß	24	Pack	9	13,67	26,99	ermässigt	78
9814	Gebäck	2	kg	9	13,56	34,99	ermässigt	46

Abbildung C.11
Tabelle Bestellung
(ohne Timestamp)

mid	bnr	kid	bestelldatum
1	33	1	20.11.2007
1	4711	2	19.11.2007
1	4714	6	19.11.2007
1	4856	2	20.11.2007
1	4857	6	20.11.2007
1	4858	6	21.11.2007
1	4861	92	21.11.2007
1	4862	10	22.11.2007
1	4902	9	21.11.2007
1	4905	6	22.11.2007
1	4906	2	22.11.2007
1	4907	25	22.11.2007
1	4912	6	23.11.2007
1	4913	25	23.11.2007
1	4914	41	23.11.2007
1	4915	30	23.11.2007
1	4916	31	23.11.2007
1	4918	68	23.11.2007
1	49030	16	21.11.2007
1	49032	9	22.11.2007
2	12552	24	19.11.2007
2	12554	4	21.11.2007
2	12556	40	21.11.2007
2	12558	20	21.11.2007
3	90940	27	22.11.2007

mnr	bnr	pos	anr	anzahl
1	4711	1	1002	10
1	4711	2	9801	5
1	4714	1	2120	2
1	4714	2	9058	10
1	4714	3	9056	1
1	4856	1	6001	10
1	4857	1	9801	5
1	4857	2	1002	10
1	4858	1	6001	25
1	4861	1	1123	10
1	4861	2	2132	100
1	4861	3	6001	10
1	4862	1	1002	10
1	4902	1	1002	25
1	4902	2	9058	3
1	4902	3	9814	5
1	4902	4	9801	2
1	4902	5	9056	1
1	4905	1	5342	3
1	4905	2	5346	10
1	4905	3	4324	20
1	4906	1	6001	100
1	4907	1	5349	10
1	4912	1	9801	10
1	4912	2	9814	20
1	4912	3	9058	2
1	4912	4	5342	5
1	4912	5	4468	10
1	4912	6	4324	10
1	4913	1	5349	20
1	4914	1	9058	10
1	4915	1	2132	1
1	4915	2	2231	10
1	4915	3	4323	25
1	4916	1	9801	5
1	4916	2	9814	5
1	4918	1	6001	40
1	4918	2	4324	20
1	4918	3	4468	20
1	4918	4	5346	30
1	4918	5	5342	10
1	4918	6	4323	20
1	49030	1	1002	10
1	49030	2	4324	10
1	49032	1	9056	10
1	49032	2	9058	20
1	49032	3	1002	10
2	12552	1	1002	20
2	12552	2	2120	2
2	12554	1	6001	25
2	12554	2	1002	10
2	12554	3	4324	25
2	12556	1	6001	10
2	12558	1	6001	30
2	12558	2	9056	5
2	12558	3	5349	10
2	12558	4	5342	20
2	12558	5	4468	10
2	12558	6	5346	3
2	12558	7	2132	2
3	90940	1	2132	5
3	90940	2	6001	10
3	90940	3	5342	5

Abbildung C.12
Tabelle Bestellposition (ohne Timestamp)

Stichwortverzeichnis

Symbols

!= 110, 126
% 106, 110, 176
* 92, 106, 110
< 110, 126
<= 110, 126
<> 110
> 110, 126
>= 110, 126
? 110
_ 110
|| 180

Numerics

3-Ebenen-Modell 220

A

ABSOLUTE 175
ACID 361
ADD INDEX 371
Änderungsanomalie 242
Aggregatfunktion 138, 168, 193, 203, 295
Aktualisierungsabfrage 213
ALIAS 147
Alias 94, 96, 98, 303
  MS Access 96
  MySQL 97
  Syntax 99
  Tabellenname 97, 118
ALL 98, 111, 147, 307, 330
Alphanumerisch 151, 153, 165
ALTER DOMAIN 299
ALTER ROLE 349
ALTER TABLE 281, 371
  MySQL-Syntax 283
  Syntax 282
ALTER VIEW 294
AND 108, 147
Anforderungsanalyse 223
Anführungszeichen openBase 98
Anomalie 241, 271
ANSI 20, 95, 152
ANY 111, 307
API 381
Archiv 204
ARRAY 152
Artikeldatenbank 37
AS 94, 147
ASC 101, 147, 153
ASCENDING 101
ASCII 180
Assoziation siehe Beziehung
Atomarität 361
Attribut 225, 226
  Umsetzung ERM in Tabelle 239
Attribut siehe Datenfeld
Ausdruck 167
Ausgangszustand 38
Authentifikation 341, 342
AUTOCOMMIT 359
AUTOWERT 190
AVG 139, 143, 194

B

B-Baum 375
Bedingung 106, 145
  Klammerung 108
  Syntax 110
Bedingungsliste 109
BEGIN TRANSACTION 360
Benutzer 340
Benutzergruppe siehe Benutzerrolle
Benutzerrolle 341, 342
Benutzervariable 367
BETWEEN 108, 110
Beziehung 19, 31, 33, 116, 225, 228, 236
  1 zu 1 229
  1 zu n 229
  CREATE TABLE 263

fehlende 131
Grad der 228
Kardinalität 230
n zu n 229
rekursive 231
Umsetzung ERM in Tabelle 238
BIGINT 151, 157
BINARY 155
BIT 152, 162
BLOB 152, 162
BOOLEAN 152
Boolesche Logik 106

C

CamelCase 30, 95
CASCADE 271
CASE 191
CAST 189
CEILING 175
CHAR 154, 180
CHARACTER 151
CHARACTER SET 251
CHARACTER VARYING 151
CHECK 266, 275
CHECK OPTION 296
CLOB 152, 155
COLLATE 102, 251
COLLATION 252
COMMIT 358
COMMIT WORK 360
CONCAT 149
CONCATENATE 180
Condition-JOIN 123
CONVERT 155, 189
CORR 196
COUNT 136, 138, 143, 193, 194
COVAR_ SAMP 196
COVAR_POP 195
CREATE DATABASE 250
Firebird 255
MySQL 250
CREATE DOMAIN 152, 298
CREATE INDEX 221, 371
CREATE PROCEDURE 380
CREATE ROLE 342, 349, 353
CREATE SCHEMA 250, 255
MS Access 253
MySQL 251
openBase 256
Oracle 254
CREATE TABLE 257
mit Unterabfrage 317
Syntax 258, 266
CREATE TABLESPACE 221, 379
CREATE TRIGGER 381
CREATE TYPE 152
CREATE USER 254, 341, 353
CREATE VIEW 286
CROSS JOIN 131
CURRENCY 159
CURRENT_DATE 183
CURRENT_TIME 183
CURRENT_TIMESTAMP 170, 183

D

Data Dictionary 23
DATE 151, 162
Datenbankentwurf 222, 232
Datenbankmodellierung 219
Datenbankschema 29, 249
Datenfeld 30, 136, 149
Datengruppe 145
Datenhaltungssystem 46, 221
Datensatz 31, 136, 154, 200
ändern 210
Begrenzung 104
einfügen 199
identischer 98
löschen 215, 216
Datensatzmenge 90
Datenschutz 340
Datensicht siehe VIEW
Datentyp 31, 149, 258, 297
alphanumerisch 153, 156, 165
binäre und sonstige 162
Datum/Uhrzeit 160, 165
Dezimalzahl 159
ganzzahlig 156
Gleitkommazahl 159
numerisch 165
Datentypcode 152
DATETIME 162
Datum/Uhrzeit 165
Datum/Zeit 151
Datumsdarstellung 105
Dauerhaftigkeit 361
DCL 21, 339
DDL 21, 249
DECIMAL 151, 159
DEFAULT 200, 259

DELETE 215
mit Unterabfrage 323
Syntax 216
TRUNCATE 217
Denormalisierung 247
DESC 101, 147
DESCENDING 101
Detailtabelle 270
Dezimalzahl 159
DIN 5007 252
DISTINCT 98, 147, 295, 330
DML 21, 26
Domäne 227, 297
DOUBLE 151, 159
DROP DATABASE 253
DROP DOMAIN 300
DROP INDEX 375
DROP ROLE 342, 349, 351
DROP SCHEMA 253
DROP TABLE 284
DROP USER 342
DROP VIEW 294

E

Eigenschaft siehe Attribut
Einfügeanomalie 241
Entität 225, 226, 232
Kandidat 234
Umsetzung ERM in Tabelle 238
Entity-Relationship-Modell siehe ERM
Entwurf
konzeptioneller 223
logischer 223
Entwurfssicht 98
Equi-JOIN 125
ERM 224, 232
Ersatzname 94, 99
Escape-Zeichen 165
EXCEPT 335
EXISTS 111, 310
EXP 175
EXTRACT 183

F

Fehlbedienung 340
Feld
Datentyp 149
Name 95, 167
NULL 163
Feld siehe Datenfeld
Felddefinition 258, 264
Feldname 258
Feldnamensliste 92
Firebird 74
FIRST 143, 194
FLOAT 151, 159
FLOOR 175
FOREIGN KEY 269
Fremdschlüssel 34, 113, 115, 133
CREATE TABLE 268
Erstellung 264
Fremdschlüsseltabelle 270
FROM 99, 117, 136, 147
FULL OUTER JOIN 130
Funktion 136, 153
Aggregatfunktion 138, 168, 193
alphanumerisch 169, 180
Casting 169, 187
Datum 169
Datum/Uhrzeit 183
MS Access 172
NULL 168
numerisch 169, 175
Skalarfunktion 168
sonstige 169
VBA 172
Funktionale Abhängigkeit 244

G

Gleitkommazahl 158
GMT 161
GRANT 254, 342, 343, 353
GREATEST 176
Großschreibung 95
GROUP BY 134, 147
Syntax 136
VIEW 295
Gruppendatensatz 136, 145
Gruppierung 136
einstufig 134
mehrstufig 136
MS Access 139
openBase 139
totale 140
Gruppierungsfeld 134
gsec 350

H

HAVING 144, 147, 203
  Syntax 145
  VIEW 295
HSQL 83

I

IDENTIFIED BY 341
Identifikation 342
IF 150, 190
Implementierung 224
IN 108, 309
Index 369, 378
INNER JOIN 116
InnoDB 46, 360
INSERT 199
  mit Unterabfrage 202, 321
  Syntax 201, 203, 205
Installation
  Firebird 75
  MySQL 41
  openBase 83
  Oracle 69
Installation Beispiel
  Firebird 79
  MS Access 66
  MySQL 58
  openBase 84
  Oracle 72
INT 157
INTEGER 151, 156, 157
Integritätsbedingung 266
INTERSECT 333
INTERVAL DAY 151
INTERVAL HOUR 151
Intervall 151
IS NULL 110, 164
Isolation 361
Isolationslevel 364

J

Jahrtausendwechsel 160
JOIN 114, 126, 130, 225
  alte Syntax 117
  Condition-JOIN 123
  CROSS JOIN 131
  Equi-JOIN 125
  INNER JOIN 116
  mehrere Tabellen 119
  NATURAL JOIN 124, 134
  Non-Equi-JOIN 125
  OUTER JOIN 127
  rekursiver 117
  Self-JOIN 118
  Syntax 116
  über mehrere Felder 133
  USING 134
  USING JOIN 123
  VIEW 289

K

Kardinalität 230
Kartesisches Produkt 131
Klammer
  eckige 90
  geschweifte 90
Klammersetzung 122
Klammerung 108
Konsistenz 361
Konto 340
Kursdatenbank 35

L

LAST 143, 194
LEAST 175
Leerzeichen 94
LEFT OUTER JOIN 127
LENGTH 180
LIKE 110
LIMIT 213
Literal 164, 167
LN 175
Löschabfrage 217
Löschanomalie 242
Lösung 35
LOG 175
LOWER 180
LTRIM 181

M

Mandantenfähigkeit 133
Mastertabelle 269
MAX 143, 194
MDB 66
MEDIUMINT 157
Memo 155
Metadaten 249
MIN 143, 194

MINUS 334
MOD 174, 176
Modell 220, 224
Modellierung 219
MS Access 65
MULTISET 152
MyISAM 46, 360
MySQL 41
mysql 53
MySQL Query Browser 58
mysqladmin 53

N

NATURAL JOIN
  über mehrere Felder 134
  USING JOIN 124
NCHAR 156
NO ACTION 271
Non-Equi-JOIN 125
Normalform 245
Normalisierung 243
NOT 108, 110
NOT EXISTS 259
NULL 129, 163, 200, 259
  OUTER JOIN 129
NUMBER 158
NUMERIC 151, 159
Numerisch 151, 165
NVARCHAR 156

O

Oberfläche 21
  Firebird 77
  MySQL 54
  Oracle 71
  Prinzip 21
ODBC-Modus 172
OLE 162
ON 116
  Condition-JOIN 123
ON DELETE 269
ON UPDATE 269
openBase 82
openOffice.orgbase 82
openOffice.orgbase siehe openBase
Operator 110
OR 108, 147
Oracle 69
ORDER BY 100, 147, 329
  Syntax 103
OUTER JOIN 127
  FULL OUTER JOIN 130
  LEFT OUTER JOIN 127
  MS Access 128
  RIGHT OUTER JOIN 129

P

Parameter 169
Parser 24
Partitionierung 380
Passwort 47
Performance 104, 122, 146, 154, 247
PHP 382
PI 176
Platzhalter 106, 375
  % 111
  * 92
  _ 111
POSITION 180
POWER 175
Präfix 30
Präfixlänge 371
Primärschlüssel 32, 113, 115, 227, 234, 260
  CREATE TABLE 266
  künstliche 33
  mehrere 34, 133
  sprechend 32
Primärschlüsseltabelle 269
PRIMARY KEY 267
Primary key siehe Primärschlüssel
Privileg 340
Programmierschnittstelle 24, 152
Programmiersprache 143
Programmierung 305
Projektion 106
Prozedur 380

Q

Qualifikation 291
Qualifizierung 93, 120

R

RANDOM 176
READ COMMITED 365
READ UNCOMMITTED 364
REAL 151, 159
Rechte 343
Recoverymanager 25, 357
Redundanz 240
REFERENCES 264

Referenzielle Integrität 270, 272, 273, 284
Reguläre Ausdrücke 111
Reihenfolge 91, 100
  INSERT 199
  JOIN 121
  Sortierung Feldnamen 103
Relation 19
RENAME USER 342
REPEATABLE READ 366
REPLACE 180
Reportgenerator 143
REVOKE 344
RIGHT OUTER JOIN 129
ROLE 341
ROLLBACK 359
ROLLBACK TRANSACTION 360
ROLLBACK WORK 360
Rollenkonzept 347
root 47
ROUND 176
RTRIM 181
Rückgabewert 169
Rückgängig 211

S

Schema
  externes 221, 285
  internes 221
  logisches 221
Schlüssel 226
  künstlicher 227
  Schlüsselkandidat 227
Schlüssel siehe Primärschlüssel
Schlüsselkandidat 227
Schmutziges Lesen 364
Schnappschuss 318
Schnelleinstieg
  Firebird 39
  MS Access 40
  MySQL 38
  openBase 40
  Oracle 38
Schreibweise
  gemischte 94, 97
SELECT 89, 288
  ALL 98
  DISTINCT 98
  einfache 91
  GROUP BY 134
  HAVING 144
  JOIN 115
  ORDER BY 103
  Sortierung 100
  Syntax 147
  WHERE 105, 142
Selektion 106
Self-JOIN 118
SERIALIZABLE 367
SET 205, 213
SET DEFAULT 272
SET NULL 272
SET PASSWORD 347
SET TRANSACTION 364
SIGN 176
SIGNED 157
SINGLE 159
Skalarfunktion 168
SMALLINT 151, 157
Sommerzeit 161
Sonderzeichen 94
Sortierung 100, 252
  alphanumerisch 102
  NULL 103
  Position 103
  Test 104
Speicherplatz 153, 154, 157
Speicherverwaltung 25
Sperre 361
  auf Satzebene 362
  auf Tabellenebene 362
Sperren 213
Sperrstrategie 362
SQL
  DCL 21, 339
  DDL 21, 249
  Definition 15
  Dialekt 13
  DML 21, 26
  Geschichte 19
  Grundregeln 28
  Grundstruktur 27
SQRT 176
Standardisierung 20
Standardwert siehe DEFAULT
START TRANSACTION 358
STDDEV 143
STDDEV_POP 195
STDDEV_SAMP 195

Strich senkrechter 98
Sub-Query siehe Unterabfrage
Sub-SELECT siehe Unterabfrage
SUBSTRING 180
SUM 139, 143, 194
Superuser 47, 340, 345
SYSTEM 70, 347
System.mdw 352

T

Tabelle 29
  mandantenfähige 133
  Name 30, 97
  Sicht auf 285
  virtuelle 90
Tabellenerstellungsabfrage 262
Tablespace 379
TEXT 155
TIME 151, 162
TIMESTAMP 151, 161, 167, 261
TIMEZONE 161
TINYINT 157
Transaktion 255, 358, 360
Transaktionsmanager 25, 357
Trigger 380
TRIM 180
TRUNCATE 176, 217
Tupel siehe Datensatz

U

UDF 171
Unicode 153
UNION 295, 328, 330
UNIQUE 99, 260, 278
UNSIGNED 157
Unterabfrage 202, 301
  CREATE TABLE 317
  in der FROM-Klausel 313
  korrelierte 312
  mit ALL/ANY 307
  mit IN/EXISTS 309
  mit Vergleichsoperator 305
  MS Access 315
  synchronisierte 312
  Syntax 304
  verbundene 312
  VIEW 295
Unterstrich 96
UPDATE 210
  mit Unterabfrage 320
  Syntax 212
UPPER 181
USING
  einfach 123
  über mehrere Felder 134
USING JOIN 123
UTC 161

V

VALUES 201
VAR 143
VAR_POP 195
VAR_SAMP 195
VARBINARY 155
VARCHAR 154
VBA-Funktionen 172
VIEW 221, 285
  änderbar 294
  als Abfrage 293
  MERGE 292
  updateable 294
  Zugriffsschutz 354

W

WHERE 105, 126, 142, 146, 147, 155, 203
  Klausel 109
  Syntax 112
WIDTH_BUCKET 179, 314
Wildcard 111
WITH GRANT OPTION 344

Z

Zeichensatz 102, 155
Zeitstempel 161, 261
ZEROFILL 157